

19th International Symposium on Parameterized and Exact Computation

IPEC 2024, September 4–6, 2024, Royal Holloway, University of
London, Egham, United Kingdom

Edited by

Édouard Bonnet

Paweł Rzążewski



Editors

Édouard Bonnet 

LIP, ENS Lyon, France
edouard.bonnet@ens-lyon.fr

Paweł Rządewski 

Warsaw University of Technology, Poland
University of Warsaw, Poland
pawel.rzadewski@pw.edu.pl

ACM Classification 2012

Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Approximation algorithms analysis; Theory of computation → Graph algorithms analysis; Theory of computation → Algorithm design techniques; Theory of computation → Mathematical optimization

ISBN 978-3-95977-353-9

Published online and open access by

Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at <https://www.dagstuhl.de/dagpub/978-3-95977-353-9>.

Publication date

December, 2024

Bibliographic information published by the Deutsche Nationalbibliothek

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

License

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



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

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

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/LIPIcs.IPEC.2024.0

ISBN 978-3-95977-353-9

ISSN 1868-8969

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

LIPICs – Leibniz International Proceedings in Informatics

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

Editorial Board

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

ISSN 1868-8969

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

■ Contents

Preface	
<i>Édouard Bonnet and Paweł Rzażewski</i>	0:ix
Conference Organization	
.....	0:xi
The Tradition of I(W)PEC	
.....	0:xiii
List of Authors	
.....	0:xv

Regular Papers

Combining Crown Structures for Vulnerability Measures	
<i>Katrin Casel, Tobias Friedrich, Aikaterini Niklanovits, Kirill Simonov, and Ziena Zeif</i>	1:1–1:15
Linear-Time MaxCut in Multigraphs Parameterized Above the Poljak-Turzík Bound	
<i>Jonas Lill, Kalina Petrova, and Simon Weber</i>	2:1–2:19
Twin-Width Meets Feedback Edges and Vertex Integrity	
<i>Jakub Balabán, Robert Ganian, and Mathis Rocton</i>	3:1–3:22
On the Parameterized Complexity of Eulerian Strong Component Arc Deletion	
<i>Václav Blažej, Satyabrata Jana, M. S. Ramanujan, and Peter Strulo</i>	4:1–4:20
Unsplittable Flow on a Short Path	
<i>Ilan Doron-Arad, Fabrizio Grandoni, and Ariel Kulik</i>	5:1–5:22
On Equivalence of Parameterized Inapproximability of k -Median, k -Max-Coverage, and 2-CSP	
<i>Karthik C. S., Euiwoong Lee, and Pasin Manurangsi</i>	6:1–6:18
On Controlling Knockout Tournaments Without Perfect Information	
<i>Václav Blažej, Sushmita Gupta, M. S. Ramanujan, and Peter Strulo</i>	7:1–7:15
Kernelization for Orthogonality Dimension	
<i>Ishay Haviv and Dror Rabinovich</i>	8:1–8:17
Fine-Grained Complexity of Multiple Domination and Dominating Patterns in Sparse Graphs	
<i>Marvin Künnemann and Mirza Redzic</i>	9:1–9:18
Subexponential Algorithms for Clique Cover on Unit Disk and Unit Ball Graphs	
<i>Tomohiro Koana, Nidhi Purohit, and Kirill Simonov</i>	10:1–10:17
Solving Co-Path/Cycle Packing and Co-Path Packing Faster Than 3^k	
<i>Yuzi Liu and Mingyu Xiao</i>	11:1–11:17



A Polynomial Time Algorithm for Steiner Tree When Terminals Avoid a Rooted K_4 -Minor <i>Carla Groenland, Jesper Nederlof, and Tomohiro Koana</i>	12:1–12:17
Kick the Cliques <i>Gaétan Berthe, Marin Bougeret, Daniel Gonçalves, and Jean-Florent Raymond</i> ..	13:1–13:17
The Parameterized Complexity Landscape of Two-Sets Cut-Uncut <i>Matthias Bentert, Fedor V. Fomin, Fanny Hauser, and Saket Saurabh</i>	14:1–14:23
Preprocessing to Reduce the Search Space for Odd Cycle Transversal <i>Bart M. P. Jansen, Yosuke Mizutani, Blair D. Sullivan, and Ruben F. A. Verhaegh</i>	15:1–15:18
Modularity Clustering Parameterized by Max Leaf Number <i>Jaroslav Garvardt and Christian Komusiewicz</i>	16:1–16:14
Subset Feedback Vertex Set in Tournaments as Fast as Without the Subset <i>Satyabrata Jana, Lawqueen Kanesh, Madhumita Kundu, and Saket Saurabh</i>	17:1–17:17
Parameterised Distance to Local Irregularity <i>Foivos Fioravantes, Nikolaos Melissinos, and Theofilos Triommatis</i>	18:1–18:15
Single-Machine Scheduling to Minimize the Number of Tardy Jobs with Release Dates <i>Matthias Kaul, Matthias Mnich, and Hendrik Molter</i>	19:1–19:15
Quasi-Linear Distance Query Reconstruction for Graphs of Bounded Treelength <i>Paul Bastide and Carla Groenland</i>	20:1–20:11
Component Order Connectivity Admits No Polynomial Kernel Parameterized by the Distance to Subdivided Comb Graphs <i>Jakob Greilhuber and Roohani Sharma</i>	21:1–21:17
Dynamic Parameterized Feedback Problems in Tournaments <i>Anna Zych-Pawlewicz and Marek Żochowski</i>	22:1–22:15
Parameterized Shortest Path Reconfiguration <i>Nicolas Bousquet, Kshitij Gajjar, Abhiruk Lahiri, and Amer E. Mouawad</i>	23:1–23:14
Roman Hitting Functions <i>Henning Fernau and Kevin Mann</i>	24:1–24:15
Matching (Multi)Cut: Algorithms, Complexity, and Enumeration <i>Guilherme C. M. Gomes, Emanuel Juliano, Gabriel Martins, and Vinicius F. dos Santos</i>	25:1–25:15
The PACE 2024 Parameterized Algorithms and Computational Experiments Challenge: One-Sided Crossing Minimization <i>Philipp Kindermann, Fabian Klute, and Soeren Terziadis</i>	26:1–26:20

PACE Solver Descriptions

PACE Solver Description: Exact Solution of the One-Sided Crossing Minimization Problem by the MPPEG Team <i>Michael Jünger, Paul J. Jünger, Petra Mutzel, and Gerhard Reinelt</i>	27:1–27:4
--	-----------

PACE Solver Description: UzL Exact Solver for One-Sided Crossing Minimization
Max Bannach, Florian Chudigiewitsch, Kim-Manuel Klein, and Marcel Wienöbst 28:1–28:4

PACE Solver Description: CRGone
Alexander Dobler 29:1–29:4

PACE Solver Description: Crossy – An Exact Solver for One-Sided Crossing Minimization
Tobias Röhr and Kirill Simonov 30:1–30:4

PACE Solver Description: CIMAT_Team
Carlos Segura, Lázaro Lugo, Gara Miranda, and Edison David Serrano Cárdenas 31:1–31:4

PACE Solver Description: Martin_J_Geiger
Martin Josef Geiger 32:1–32:4

PACE Solver Description: Arcee
Kimon Boehmer, Lukas Lee George, Fanny Hauser, and Jesse Palarus 33:1–33:4

PACE Solver Description: LUNCH – Linear Uncrossing Heuristics
Kenneth Langedal, Matthias Bentert, Thorgal Blanco, and Pål Grønås Drange ... 34:1–34:4

PACE Solver Description: OCMu64, a Solver for One-Sided Crossing Minimization
Ragnar Groot Koerkamp and Mees de Vries 35:1–35:5

■ Preface

The 19th International Symposium on Parameterized and Exact Computation (IPEC 2024) took place at Royal Holloway, University of London in Egham, United Kingdom, on 4-6 September, 2024. It was a part of ALGO 2024.

IPEC is an annual conference covering all aspects of parameterized and exact algorithms, and complexity. To emphasize the link between theory and practice, IPEC hosts, since 2016, the Parameterized Algorithms and Computational Experiments (PACE) Challenge. This year's problem was one-sided crossing minimization, and attracted a record number of participants and teams.

IPEC 2024 had two keynote talks. The first was an invited tutorial titled *Structurally tractable graph classes*, given by Szymon Toruńczyk. The second was by Daniel Lokshtanov, one of the 2024 EATCS-IPEC Nerode Prize winners. Indeed, the prize was awarded to Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos for their paper *(Meta) Kernelization* [FOCS 2010, JACM 2016].

We received a total of 51 submissions, out of which the Program Committee accepted 25. The prize for the Best Paper was awarded to Katrin Casel, Tobias Friedrich, Aikaterini Niklanovits, Kirill Simonov, and Ziena Zeif for their paper *Combining Crown Structures for Vulnerability Measures*. The extended abstract of all accepted papers, as well as short descriptions of the best solutions to the PACE Challenge, are included in this volume.

Many individuals contributed to the success of IPEC 2024. In particular, we thank:

- All authors who submitted their results to IPEC,
- The members of the PC and subreviewers, who graciously gave their time and energy,
- The Steering Committee for their helpful guidance,
- The invited speakers and the 2024 EATCS-IPEC Nerode Prize Committee: Thore Husfeld, Sang-il Oum, and Russell Impagliazzo,
- ALGO 2024 organizing committee, chaired by Argyrios Deligkas and Eduard Eiben, and
- The session chairs and all other participants.

September 2024

Édouard Bonnet
Paweł Rzażewski



■ Conference Organization

Program Committee

Édouard Bonnet (ENS Lyon, LIP, France, co-chair)
Nick Brettell (Victoria University of Wellington, New Zealand)
David Eppstein (University of California, Irvine, USA)
Piotr Faliszewski (AGH University, Kraków, Poland)
Andreas Emil Feldmann (University of Sheffield, UK)
Jacob Focke (CISPA Helmholtz Center for Information Security, Saarbrücken, Germany)
Panos Giannopoulos (City University, London, UK)
Petr Hliněný (Masaryk University, Brno, Czech Republic)
Lars Jaffke (University of Bergen, Norway)
Petteri Kaski (Aalto University, Finland)
Eunjung Kim (CNRS, LAMSADE, Paris-Dauphine University, France)
Tuukka Korhonen (University of Bergen, Norway)
Stephan Kreutzer (TU Berlin, Germany)
Paloma T. Lima (IT University of Copenhagen, Denmark)
Karolina Okrasa (Warsaw University of Technology, Poland)
Anthony Perez (LIFO, University of Orléans, France)
Paweł Rzażewski (Warsaw University of Technology & Univ. of Warsaw, Poland, co-chair)
Ignasi Sau (LIRMM, CNRS, Université de Montpellier, France)
Darren Strash (Hamilton College, Clinton, USA)
Ryan Williams (Massachusetts Institute of Technology, Cambridge, USA)

Additional Reviewers

Cornelius Brand	William Lochet
Eric Brandwein	Raul Lopes
Guilherme de Castro Mendes Gomes	Tobias Mömke
Oscar Defrain	Laure Morelle
Pål Grønås Drange	Jan Obdržálek
Henning Fernau	Fiona Skerman
Harmender Gahlawat	Andreas Wiese
Bingkai Lin	Michał Włodarczyk

Steering Committee

Akanksha Agrawal (2024–27)
Hans Bodlaender (2023–26)
Édouard Bonnet (2023–2026)
Holger Dell (2021–2024)
Fedor Fomin (2021–24)
Łukasz Kowalik (2022–25)
Erik Jan van Leeuwen (2024–27)
Neeldhara Misra (2022–25)
Jesper Nederlof (2021–24)
Paweł Rzażewski (2023–26)
Dimitrios Thilikos (2024–27)
Magnus Wahlström (2022–25)

19th International Symposium on Parameterized and Exact Computation (IPEC 2024).

Editors: Édouard Bonnet and Paweł Rzażewski




Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

■ The Tradition of I(W)PEC

2004	1st IWPEC,	Bergen, Norway	chairs: Rodney Downey, Michael Fellows, Frank Dehne
2006	2nd IWPEC,	Zürich, Switzerland	chairs: Hans L. Bodlaender, Michael A. Langston
2008	3rd IWPEC,	Victoria, Canada	chairs: Martin Grohe, Rolf Niedermeier
2009	4th IWPEC,	Copenhagen, Denmark	chairs: Jianer Chen, Fedor V. Fomin
2010	5th IPEC,	Chennai, India	chairs: Venkatesh Raman, Saket Saurabh
2011	6th IPEC,	Saarbrücken, Germany	chairs: Dániel Marx, Peter Rossmanith
2012	7th IPEC,	Ljubljana, Slovenia	chairs: Dimitrios Thilikos, Gerhard Woeginger
2013	8th IPEC,	Sophia Antipolis, France	chairs: Gregory Gutin, Stefan Szeider
2014	9th IPEC,	Wrocław, Poland	chairs: Marek Cygan, Pinar Heggernes
2015	10th IPEC,	Patras, Greece	chairs: Thore Husfeldt, Iyad Kanj
2016	11th IPEC,	Arrhus, Denmark	chairs: Jiong Guo, Danny Hermelin
2017	12th IPEC,	Vienna, Austria	chairs: Daniel Lokshtanov, Naomi Nishimura
2018	13th IPEC,	Helsinki, Finland	chairs: Christophe Paul, Michał Pilipczuk
2019	14th IPEC,	Munich, Germany	chairs: Bart Jansen, Jan Arne Telle
2020	15th IPEC,	Hong Kong virtual event	chairs: Yixin Cao, Marcin Pilipczuk
2021	16th IPEC,	Lisbon, Portugal virtual event	chairs: Petr Golovach, Meirav Zehavi
2022	17th IPEC,	Potsdam, Germany	chairs: Holger Dell, Jesper Nederlof
2023	18th IPEC,	Amsterdam, the Netherlands	chairs: Neeldhara Misra, Magnus Wahlström
2024	19th IPEC,	Egham, United Kingdom	chairs: Édouard Bonnet and Paweł Rzażewski




■ List of Authors

Jakub Balabán  (3)
Faculty of Informatics, Masaryk University,
Brno, Czech Republic


Max Bannach  (28)
European Space Agency, Noordwijk,
The Netherlands

Paul Bastide  (20)
LaBRI - Université de Bordeaux, France;
TU Delft, The Netherlands


Matthias Bentert (14, 34)
University of Bergen, Norway


Gaétan Berthe  (13)
LIRMM, Université de Montpellier, CNRS,
Montpellier, France


Thorgal Blanco  (34)
University of Bergen, Norway


Václav Blažej  (4, 7)
University of Warwick, Coventry, UK


Kimion Boehmer (33)
Université Paris-Saclay, France

Marin Bougeret  (13)
LIRMM, Université de Montpellier, CNRS,
Montpellier, France

Nicolas Bousquet  (23)
Univ. Lyon, LIRIS, CNRS, Université Claude
Bernard Lyon 1, Villeurbanne, France

Guilherme C. M. Gomes  (25)
Departamento de Ciência da Computação,
Universidade Federal de Minas Gerais, Belo
Horizonte, Brazil;
CNRS/LIRMM, Montpellier, France

Katrin Casel  (1)
Humboldt Universität zu Berlin, Germany


Florian Chudigiewitsch  (28)
Institute for Theoretical Computer Science,
University of Lübeck, Germany


Mees de Vries  (35)
Unaffiliated, The Netherlands


Alexander Dobler  (29)
TU Wien, Austria

Ilan Doron-Arad (5)
Computer Science Department,
Technion, Haifa, Israel


Pål Grønås Drange  (34)
University of Bergen, Norway

Vinicius F. dos Santos  (25)
Departamento de Ciência da Computação,
Universidade Federal de Minas Gerais,
Belo Horizonte, Brazil


Henning Fernau  (24)
Fachbereich IV, Informatikwissenschaften,
Universität Trier, Germany


Foivos Fioravantes  (18)
Department of Theoretical Computer Science,
FIT, Czech Technical University in Prague,
Czech Republic

Fedor V. Fomin  (14)
University of Bergen, Norway

Tobias Friedrich  (1)
Hasso Plattner Institute,
University of Potsdam, Germany


Kshitij Gajjar (23)
Centre for Security, Theory & Algorithmic
Research (CSTAR), International Institute of
Information Technology, Hyderabad (IIIT-H),
India

Robert Ganian  (3)
Algorithms and Complexity Group,
TU Wien, Vienna, Austria


Jaroslav Garvardt  (16)
Institute of Computer Science, Friedrich Schiller
University Jena, Germany

Martin Josef Geiger  (32)
University of the Federal Armed Forces
Hamburg, Germany

Lukas Lee George  (33)
Technical University Berlin, Germany

Daniel Gonçalves  (13)
LIRMM, Université de Montpellier, CNRS,
Montpellier, France

Fabrizio Grandoni (5)
IDSIA, USI-SUPSI, Lugano, Switzerland

Jakob Greilhuber  (21)
TU Wien, Austria; CISPA Helmholtz Center for
Information Security, Saarbrücken, Germany;
Saarbrücken Graduate School of Computer
Science, Germany

19th International Symposium on Parameterized and Exact Computation (IPEC 2024).

Editors: Édouard Bonnet and Paweł Rzażewski



Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- Carla Groenland  (12, 20)
Delft Institute of Applied Mathematics,
The Netherlands
- Ragnar Groot Koerkamp  (35)
ETH Zurich, Switzerland
- Sushmita Gupta  (7)
The Institute of Mathematical Sciences,
HBNI, Chennai, India
- Fanny Hauser (14, 33)
Technische Universität Berlin, Germany;
University of Bergen, Norway
- Ishay Haviv  (8)
School of Computer Science, The Academic
College of Tel Aviv-Yaffo, Israel
- Satyabrata Jana  (4, 17)
University of Warwick, Coventry, UK
- Bart M. P. Jansen  (15)
Eindhoven University of Technology,
The Netherlands
- Emanuel Juliano  (25)
Departamento de Ciência da Computação,
Universidade Federal de Minas Gerais,
Belo Horizonte, Brazil
- Michael Jünger  (27)
University of Cologne, Germany
- Paul J. Jünger  (27)
University of Bonn, Germany
- Lawqueen Kanesh  (17)
Indian Institute of Technology, Jodhpur, India
- Karthik C. S.  (6)
Rutgers University, Piscataway, NJ, USA
- Matthias Kaul  (19)
Universität Bonn, Bonn, Germany
- Philipp Kindermann  (26)
Trier University, Germany
- Kim-Manuel Klein  (28)
Institute for Theoretical Computer Science,
University of Lübeck, Germany
- Fabian Klute  (26)
Polytechnic University of Catalonia,
Barcelona, Spain
- Tomohiro Koana  (10, 12)
Utrecht University, The Netherlands
- Christian Komusiewicz  (16)
Institute of Computer Science, Friedrich Schiller
University Jena, Germany
- Ariel Kulik  (5)
Computer Science Department,
Technion, Haifa, Israel
- Madhumita Kundu  (17)
University of Bergen, Norway
- Marvin Künnemann  (9)
Karlsruhe Institute of Technology, Germany
- Abhiruk Lahiri  (23)
Department of Computer Science, Heinrich
Heine University, Düsseldorf, Germany
- Kenneth Langedal  (34)
University of Bergen, Norway
- Euiwoong Lee  (6)
University of Michigan, Ann Arbor, MI, USA
- Jonas Lill (2)
Department of Computer Science,
ETH Zürich, Switzerland
- Yuxi Liu (11)
University of Electronic Science and Technology
of China, Chengdu, China
- Lázaro Lugo  (31)
Area de Computación, Centro de Investigación
en Matemáticas (CIMAT), Guanajuato, Mexico
- Kevin Mann  (24)
Fachbereich IV, Informatikwissenschaften,
Universität Trier, Germany
- Pasin Manurangsi  (6)
Google Research, Bangkok, Thailand
- Gabriel Martins  (25)
Departamento de Ciência da Computação,
Universidade Federal de Minas Gerais,
Belo Horizonte, Brazil
- Nikolaos Melissinos  (18)
Department of Theoretical Computer Science,
FIT, Czech Technical University in Prague,
Czech Republic
- Gara Miranda  (31)
Departamento de Ingeniería Informática y de
Sistemas, Universidad de La Laguna, Spain
- Yosuke Mizutani  (15)
School of Computing, University of Utah,
Salt Lake City, UT, USA
- Matthias Mnich  (19)
Hamburg University of Technology, Institute for
Algorithms and Complexity, Hamburg, Germany

- Hendrik Molter  (19)
Department of Computer Science, Ben-Gurion
University of the Negev, Beer-Sheva, Israel
- Amer E. Mouawad (23)
Department of Computer Science,
American University of Beirut, Lebanon
- Petra Mutzel  (27)
University of Bonn, Germany
- Jesper Nederlof  (12)
Department of Information and Computing
Sciences, Utrecht University, The Netherlands
- Aikaterini Niklanovits  (1)
Hasso Plattner Institute,
University of Potsdam, Germany
- Jesse Palarus  (33)
Technical University Berlin, Germany
- Kalina Petrova  (2)
Institute of Science and Technology Austria
(ISTA), Klosterneuburg, Austria
- Nidhi Purohit  (10)
National University of Singapore, Singapore
- Dror Rabinovich (8)
School of Computer Science, The Academic
College of Tel Aviv-Yaffo, Israel
- M. S. Ramanujan  (4, 7)
University of Warwick, Coventry, UK
- Jean-Florent Raymond  (13)
Univ. Lyon, CNRS, ENS de Lyon, Université
Claude Bernard Lyon 1, LIP UMR5668, Lyon,
France
- Mirza Redzic  (9)
Karlsruhe Institute of Technology, Germany
- Gerhard Reinelt  (27)
Heidelberg University, Germany
- Mathis Rocton  (3)
Algorithms and Complexity Group,
TU Wien, Vienna, Austria
- Tobias Röhr (30)
Hasso Plattner Institute,
University of Potsdam, Germany
- Saket Saurabh  (14, 17)
The Institute of Mathematical Sciences,
Chennai, India; University of Bergen, Norway
- Carlos Segura  (31)
Area de Computación, Centro de Investigación
en Matemáticas (CIMAT), Guanajuato, Mexico
- Edison David Serrano Cárdenas  (31)
Area de Matemáticas Aplicadas, Centro de
Investigación en Matemáticas (CIMAT),
Guanajuato, Mexico
- Roohani Sharma  (21)
Department of Informatics,
University of Bergen, Norway
- Kirill Simonov  (1, 10, 30)
Hasso Plattner Institute,
University of Potsdam, Germany
- Peter Strulo  (4, 7)
University of Warwick, Coventry, UK
- Blair D. Sullivan  (15)
School of Computing, University of Utah,
Salt Lake City, UT, USA
- Soeren Terziadis  (26)
Eindhoven University of Technology,
The Netherlands
- Theofilos Triommatis  (18)
School of Electrical Engineering, Electronics and
Computer Science University of Liverpool, UK
- Ruben F. A. Verhaegh  (15)
Eindhoven University of Technology,
The Netherlands
- Simon Weber  (2)
Department of Computer Science,
ETH Zürich, Switzerland
- Marcel Wienöbst  (28)
Institute for Theoretical Computer Science,
University of Lübeck, Germany
- Mingyu Xiao  (11)
University of Electronic Science and Technology
of China, Chengdu, China
- Ziena Zeif  (1)
Hasso Plattner Institute,
University of Potsdam, Germany
- Anna Zych-Pawlewicz  (22)
Institute of Informatics,
University of Warsaw, Poland
- Marek Żochowski (22)
Institute of Informatics,
University of Warsaw, Poland

Combining Crown Structures for Vulnerability Measures

Katrin Casel  


Humboldt Universität zu Berlin, Germany

Tobias Friedrich  

Hasso Plattner Institute, University of Potsdam, Germany

Aikaterini Niklanovits  

Hasso Plattner Institute, University of Potsdam, Germany

Kirill Simonov  

Hasso Plattner Institute, University of Potsdam, Germany

Ziena Zeif  

Hasso Plattner Institute, University of Potsdam, Germany

Abstract

Over the past decades, various metrics have emerged in graph theory to grasp the complex nature of network vulnerability. In this paper, we study two specific measures: (weighted) vertex integrity (wVI) and (weighted) component order connectivity (wCOC). These measures not only evaluate the number of vertices that need to be removed to decompose a graph into fragments, but also take into account the size of the largest remaining component. The main focus of our paper is on kernelization algorithms tailored to both measures. We capitalize on the structural attributes inherent in different crown decompositions, strategically combining them to introduce novel kernelization algorithms that advance the current state of the field. In particular, we extend the scope of the balanced crown decomposition provided by Casel et al. [5] and expand the applicability of crown decomposition techniques.

In summary, we improve the vertex kernel of VI from p^3 to $3p^2$, and of wVI from p^3 to $3(p^2 + p^{1.5}p_\ell)$, where $p_\ell < p$ represents the weight of the heaviest component after removing a solution. For wCOC we improve the vertex kernel from $\mathcal{O}(k^2W + kW^2)$ to $3\mu(k + \sqrt{\mu}W)$, where $\mu = \max(k, W)$. We also give a combinatorial algorithm that provides a $2kW$ vertex kernel in fixed-parameter tractable time when parameterized by r , where $r \leq k$ is the size of a maximum $(W + 1)$ -packing. We further show that the algorithm computing the $2kW$ vertex kernel for COC can be transformed into a polynomial algorithm for two special cases, namely when $W = 1$, which corresponds to the well-known vertex cover problem, and for claw-free graphs. In particular, we show a new way to obtain a $2k$ vertex kernel (or to obtain a 2-approximation) for the vertex cover problem by only using crown structures.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Crown Decomposition, Kernelization, Vertex Integrity, Component Order Connectivity

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.1

Related Version *Full Version:* <https://doi.org/10.48550/arXiv.2405.02378> [6]

1 Introduction

In the study of graph theory different scales have emerged over the past decades to capture the complex nature of network vulnerability. While the main focus is on the connectivity of vertices and edges, subtle aspects of vulnerability such as the number of resulting components, the size distribution of the remaining components, and the disparity between them are becoming increasingly interesting [3, 13, 14, 16, 15, 20]. Our focus in this study is on two



© Katrin Casel, Tobias Friedrich, Aikaterini Niklanovits, Kirill Simonov, and Ziena Zeif; licensed under Creative Commons License CC-BY 4.0

19th International Symposium on Parameterized and Exact Computation (IPEC 2024).

Editors: Édouard Bonnet and Paweł Rzażewski; Article No. 1; pp. 1:1–1:15



Leibniz International Proceedings in Informatics

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

specific ways to measure vulnerability: (weighted) vertex integrity and (weighted) component order connectivity. These measures not only evaluate the number of vertices that need to be removed to decompose a graph into fragments, but also take into account the size of the largest remaining component. Incorporating these aspects provides a more comprehensive understanding of network resilience.

Informally, *vertex integrity* (VI) is a model for the right balance between removing few vertices and keeping small connected parts of a graph. More formally, given a graph $G = (V, E)$ and a number $p \in \mathbb{N}$, the task for VI is to find a set of vertices $S \subseteq V$ such that $|S|$ plus the size of the largest component when removing S from G is at most p . In the vertex weighted version (wVI), the goal is bounding the total weight of the removed vertices plus the weight of the heaviest component by p . This problem was introduced by Barefoot et al. [2] as a way to measure vulnerability of communication networks. Recently, it has drawn the interest of the parameterized complexity community due to its status as a natural parameter that renders numerous NP-hard problems amenable to fixed parameter tractability (FPT). This means that for these problems, solutions can be computed within a time frame represented by $f(p) \cdot n^{\mathcal{O}(1)}$, where f is a computable function [20]. It is interesting to see how vertex integrity relates to other well-known measures of network structure. It imposes greater constraints compared to metrics such as treedepth, treewidth, or pathwidth, as the vertex integrity of a graph serves as an upper bound for these parameters. However, it encompasses a wider range of scenarios compared to vertex cover, where a vertex cover of a graph is an upper bound for its vertex integrity. This makes it a key in understanding how to efficiently solve problems in the world of network analysis.

The measure *component order connectivity* (COC) can be seen as the refined version of VI. Given a graph $G = (V, E)$ and two parameters $k, W \in \mathbb{N}$, the goal of COC is to remove k vertices such that each connected component in the resulting graph has at most W vertices – also known in the literature as the *W -separator problem* or *α -balanced separator problem*, where $\alpha \in (0, 1)$ and $W = \alpha|V|$. In the vertex-weighted version (wCOC), the goal is to remove vertices of total weight at most k such that the weight of the heaviest remaining component is at most W . An equivalent view of this problem is to search for the minimum number of vertices required to cover or hit every connected subgraph of size $W + 1$. In particular, $W = 1$ corresponds to covering all edges, showing that the COC is a natural generalization of the vertex cover problem.

The focus of the paper is on kernelization algorithms tailored for both weighted and unweighted versions of VI and COC when parameterized by p and $k + W$, respectively. Kernelization algorithms can be thought of as formalized preprocessing techniques aimed to reduce optimization problems. Of particular interest in this work are crown decompositions, which are generally used as established structures for safe instance reduction – where “safe” in this case means that any optimal solution to the reduced instance can efficiently be transformed into an optimal solution of the original. Essentially, a crown decomposition partitions the vertex set into distinct components: crown, head, and body. Here, the head acts as a separator between the crown and the body. This structural arrangement becomes useful when specific relationships between the head and the crown are required. Such relationships ultimately enable us to shrink instances by eliminating these designated parts from the graph. The properties of this structural layout, coupled with its existence depending on the instance size, enable the development of efficient kernelization algorithms for different problem domains. Notably, crown decompositions have also recently found utility in approximation algorithms for graph packing and partitioning problems [5]. For further exploration of crown decompositions, including their variations and applications, we recommend the comprehensive survey paper by Jacob et al. [17].

Our methods take advantage of the structural characteristics found in various crown decompositions, leading to the development of new kernelization algorithms that improve the state of the art. In essence, this work expands upon the applications of the balanced crown decomposition introduced by Casel et al. [5], specifically by integrating different crown decompositions into this framework.

Related Work

Recently, the vertex integrity problem (VI) was extensively studied as a structural graph parameter [3, 13, 15, 20]. Gima et al. [14] conducted a systematic investigation into structural parameterizations of computing the VI and wVI which was further extended by Hanaka et al. [16]. Additionally, there are notable results concerning special graph classes [1, 8, 10, 18, 23]. In our context, regarding related work on VI and wVI, Fellows and Stueckle presented an algorithm that solves the problem in $\mathcal{O}(p^{3p}n)$ [11]. This was subsequently improved by Drange et al. [10], even for the weighted case, to $\mathcal{O}(p^{p+1}n)$. In the same paper, they presented the first vertex-kernel of size p^3 for both VI and wVI.

Considering, COC and wCOC, it is unlikely that kernelization algorithms for these problems can be achieved by considering k or W alone in polynomial time. Indeed, $W = 1$ corresponds to the NP-hard vertex cover problem, which shows that W (alone) is not a suitable parameter. For the parameter k , the problem is $W[1]$ -hard even when restricted to split graphs [10]. These lower bounds lead to the study of parameterization by $k + W$. The best known algorithm with respect to these parameters finds a solution in time $n^{\mathcal{O}(1)} \cdot 2^{\mathcal{O}(\log(W) \cdot k)}$ [10]. Unless the exponential time hypothesis fails, the authors prove that this running time is tight in the sense that there is no algorithm that solves the problem in time $n^{\mathcal{O}(1)} \cdot 2^{\mathcal{O}(\log(W) \cdot \text{OPT})}$. The best known approximation algorithm has a multiplicative gap guarantee of $\mathcal{O}(\log(W))$ to the optimal solution with a running time of $n^{\mathcal{O}(1)} \cdot 2^{\mathcal{O}(W)}$ [21]. In [21], they also showed that the superpolynomial dependence on W may be needed to achieve a polylogarithmic approximation. Using this algorithm as a subroutine, the vertex integrity can be approximated within a factor of $\mathcal{O}(\log(\text{OPT}))$, where OPT is the vertex integrity.

Regarding kernelization algorithms, there is a sequence of results which successively improve the vertex-kernel of COC. The first results came from Chen et al. [7] and Drange et al. [10], who provided kernels of size in $\mathcal{O}(kW^3)$ and $\mathcal{O}(k^2W + kW^2)$, respectively. The result of Drange et al. also holds for wCOC and is the only result for this case. This was improved simultaneously by Xiao [24] as well as by Kumar and Lokshantov [19] to a $\mathcal{O}(kW^2)$ kernel. These works also provide the first $\mathcal{O}(kW)$ kernels, but with different constants and running times. Kumar and Lokshantov [19] present a $2kW$ kernel in a running time of $n^{\mathcal{O}(W)}$ by using linear programming (LP) methods with an exponential number of constraints. The runtime can be improved to $2^{\mathcal{O}(W)} \cdot n^{\mathcal{O}(1)}$ as already mentioned in the book of Fomin et al. [12] (Section 6.4.2). Roughly speaking, the idea is to use the ellipsoid method with separation oracles to solve the linear program, where the separation oracle uses a method called color coding to find violated constraints that makes it polynomial in W . Note that if W is a constant this $2kW$ kernel is a polynomial time kernel improving on some previous results. This includes for instance the improvement of the $5k$ kernel provided by Xiao and Kou [25] to a $4k$ kernel for the well-studied P_2 -covering problem, where a P_2 is a path with 2 edges. The first linear kernel in both parameters in polynomial time, i.e. an $\mathcal{O}(kW)$ vertex kernel, is presented by Xiao [24], who provides a $9kW$ vertex kernel. Finally, this was improved by Casel et al. [5] to a $3kW$ vertex kernel, which also holds for a more general setting. Namely, to find k vertices in a vertex weighted graph such that after their removal each component

weights at most W . Note that the weights of the separator, i.e. the chosen k vertices, play no role in this problem compared to wCOC. With the exception of the $\mathcal{O}(k^2W + kW^2)$ vertex kernel of Drange et al. [10], all achieved vertex kernels essentially use crown structures.

Our Contribution

The provided running times are based on an input graph $G = (V, E)$. We improve the vertex kernel for VI from p^3 to $3p^2$ in time $\mathcal{O}(\log(p)|V|^4|E|)$. For wVI, we improve it to $3(p^2 + p^{1.5}p_\ell)$ in time $\mathcal{O}(\log(p)|V|^4|E|)$, where $p_\ell \leq p$ represents the weight of the heaviest component after removing a solution.

To better explain the results of COC and wCOC, consider the problem of a maximum λ -packing for $\lambda \in \mathbb{N}$. Given a (vertex weighted) graph G , this problem aims to maximize the number of disjoint connected subgraphs, each of size (or weight) at least λ . It is worth noting that the size of a maximum $(W + 1)$ -packing serves as a lower bound on the size of an optimal solution of COC (or wCOC), since each element in the packing must contain at least one vertex of it – in terms of linear programming, it is the dual of COC.

For wCOC we improve the vertex kernel of $\mathcal{O}(k^2W + kW^2)$ to $3\mu(k + \sqrt{\mu}W)$ in time $\mathcal{O}(r^2k|V||E|)$, where $\mu = \max(k, W)$ and $r \leq |V|$ is the size of a maximum $(W + 1)$ -packing. For the unweighted version, we provide a $2kW$ vertex kernel in an FPT-runtime of $\mathcal{O}(r^3|V||E| \cdot r^{\min(3r, k)})$, where $r \leq k$ is the size of a maximum $(W + 1)$ -packing. Comparing this result with the state of the art, disregarding the FPT-runtime aspect, we improve upon the best-known polynomial algorithm, achieving a kernel of size $3kW$ [5]. A $2kW$ vertex kernel is also presented in [19], albeit with an exponential runtime using linear programming methods, which, as mentioned, can be enhanced to an FPT-runtime regarding parameter W . In contrast, our result is entirely combinatorial and has an FPT-runtime in the parameter of a maximum $(W + 1)$ -packing $r \leq k$. It should be noted that, strictly speaking, the $2kW$ vertex kernel of [19] and our work cannot be considered a kernel, given that the runtime dependency is exponential with respect to the parameters W and k , respectively. However, for the sake of simplicity, we will refer to it as a kernel, with an explicit mention of the runtime dependency. As previously stated, note that it is unlikely that an FPT-runtime will be able to solve COC (or wCOC) when considering either W or k alone. We further show that the algorithm computing the $2kW$ vertex kernel for COC can be transformed into a polynomial algorithm for two special cases. The first case arises when $W = 1$, i.e., for the vertex cover problem. Here, we provide a new method for obtaining a vertex kernel of $2k$ (or obtaining a 2-approximation) using only crown decompositions. The second special case is for the restriction of COC to claw-free graphs. Unfortunately, we currently do not know whether COC is hard on claw-free graphs and defer this question to future work.

Regarding these special cases, until 2017, a $2k$ vertex kernel for $W = 1$ was known through crown decompositions, albeit computed using both crown decompositions and linear programming. Previously, only a $3k$ vertex kernel was known using crown decompositions alone. In 2018, Li and Zhu [22] provided a $2k$ vertex kernel solely based on crown structures. They refined the classical crown decomposition, which possessed an additional property allowing the remaining vertices of the graph to be decomposed into a matching and odd cycles after exhaustively applying the corresponding reduction rule. In contrast, our algorithm identifies the reducible structures, which must exist if the size of the input graph exceeds $2k$. Unfortunately, we were unable to transform the FPT-runtime algorithm into a polynomial-time algorithm in general. Nevertheless, we believe that our insights into the structural properties of certain crown decompositions potentially pave the way to achieving this goal.

Lastly, all missing details, i.e. omitted proofs and algorithms as well as more detailed explanations can be found in the extended arxiv version [6].

2 Preliminaries

In this section, we briefly discuss terminology related to graphs and parameterized complexity that we use in the extended abstract (the complete terminology can be found in the extended arxiv version [6]). Additionally, we introduce the crown structures that are utilized throughout this paper.

Graph and Parameterized Terminology

We use a standard terminology for graphs $G = (V, E)$. The following is not necessarily common: For sets of vertex sets $\mathcal{V} \subseteq 2^V$ we abuse notation and use $V(\mathcal{V}) = \bigcup_{Q \in \mathcal{V}} Q$ and $N(\mathcal{V}) := \left(\bigcup_{Q \in \mathcal{V}} N(U) \right) \setminus V(\mathcal{V})$. We define $\mathbb{CC}(G) \subseteq 2^V$ as the connected components of G as vertex sets. Let $G = (V, E, w: V \rightarrow \mathbb{N})$ be a vertex weighted graph. For $V' \subseteq V$ and $G' \subseteq G$ we define $w(V')$ and $w(G')$ as $\sum_{v \in V'} w(v)$ and $\sum_{v \in V(G')} w(v)$, respectively.

For improved readability, we extend the notation of the inverse function f^{-1} . Henceforth, sometimes it returns a vertex set that is the union of the corresponding vertex sets, which is always clear from the context. For instance, for a function $f: V \rightarrow V$ on the vertices of a graph $G = (V, E)$, $h \in V$ and $V' \subseteq V$ we use $f^{-1}(h)$ also for $V(f^{-1}(h))$, and $f^{-1}(V')$ for $V(f^{-1}(V'))$, respectively.

For parameterized complexity we use the standard terminology, which is also used, for example, in [9, 12]. Of particular interest in this work are kernelizations, which can be roughly described as formalized preprocessings. More formally, given an instance (I, k) parameterized by k , a polynomial algorithm is called a kernelization if it maps any (I, k) to an instance (I', k') such that (I', k') is a yes-instance if and only if (I, k) is a yes-instance, $|I'| \leq g(k)$, and $k' \leq g'(k)$ for computable functions g, g' .

Crown Decompositions

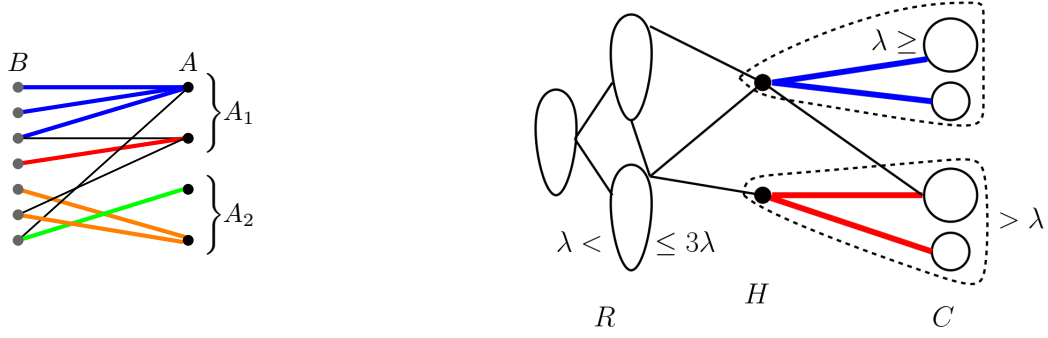
We present a more general variant of crown decomposition that also captures commonly used crown decompositions or expansions (strictly speaking the relevant parts of it), which we explain in a moment. This novel decomposition can be easily derived from existing results of [5], but to the best of our knowledge has never been used in this form before. (See Figure 1 (left) for an illustration.)

► **Definition 1** (Demanded balanced expansion and weighted crown decomposition). *For a graph $G = (A \cup B, E, w)$, a partition $A_1 \cup A_2$ of A , a function $f: \mathbb{CC}(G[B]) \rightarrow A$, demands $D = \{d_a\}_{a \in A}$ with $d_a \in \mathbb{N}$ for each $a \in A$ and $y \in \mathbb{N}$ the tuple (A_1, A_2, y, f, D) is a demanded balanced expansion (DBE) if*

1. $w(Q) \leq y$ for each $Q \in \mathbb{CC}(G[B])$
2. $f(Q) \in N(Q)$ for each $Q \in \mathbb{CC}(G[B])$
3. $N(f^{-1}(A_1)) \subseteq A_1$
4. $w(a) + w(f^{-1}(a)) \begin{cases} > d_a - y + 1 \text{ for each } a \in A_1 \\ \leq d_a + y - 1 \text{ for each } a \in A_2 \end{cases}$

To simplify the notation we introduce two further special cases of a DBE:

- If the demands are the same for each $a \in A$, e.g. $d_a = x$ for every $a \in A$ with $x \in \mathbb{N}$, then we write only the value x instead of a vector $D = \{d_a\}_{a \in A}$ in a DBE-tuple, i.e. $(A_1, A_2, y, f, D) = (A_1, A_2, y, f, x)$.



■ **Figure 1**

Left: Let $A = \{a_1, a_2, a_3, a_4\}$ be ordered in a top down manner, $w(Q) = 1$ for every $Q \in \mathbb{CC}(G[B])$ and $w(a) = 1$ for every $a \in A$. Then, $(\{a_1, a_2\}, \{a_3, a_4\}, 1, f, \{3, 1, 3, 4\})$ is a DBE, where the assignment f are depicted with corresponding colored bold edges.

Right: A λ -balanced crown decomposition, where the assignment f are depicted with corresponding colored bold edges. The two dashed lines illustrate that $w(h) + w(f^{-1}(h)) > \lambda$ for every $h \in H$ while $w(Q) \leq \lambda$ for every $Q \in \mathbb{CC}(G[C])$.

- For $q \in \mathbb{N}$ we call (C, H, f) a (q, y) crown decomposition ((q, y) -CD) if it is a $(H, \emptyset, y, f, q + y - 1)$ DBE with $H = A$ and $C = B = f^{-1}(H)$. (This term simplifies the reference to the reducible structure of crown C and head H .)

Considering a graph instance $G = (V, E)$, e.g. from VI or COC, Definition 1 usually describes only a subgraph where the reducible structure is sought, i.e. A, B are vertex subsets of V . What is crucial is that A already separates B from $V \setminus B$. Then, the vertex set $A_1 \subseteq A$ (cf. Definition 1) typically represents the structure targeted for reduction and can be seen as the “head” of a crown decomposition. The components within $\mathbb{CC}(G[B])$ assigned to A_1 (the separated part through A_1), denoted by $f^{-1}(A_1)$, form the “crown”. In the context of optimization problems such as the vertex cover problem, a successful reduction often involves the head being part of an optimal solution. Subsequently, upon its removal, the crown – separated by the head – no longer needs to be considered.

Ideally, we wish to have $A_1 = A$, but even when this is not the case the balanced part of a DBE ensures that the elements mapped to A_2 are bounded, which finally allows us to bound the number of vertices (or the sum of the vertex weights) of $A_2 \cup V(f^{-1}(A_2))$.

Consider a crown decomposition, which is a partition of the vertex set into body, head and crown, where the head separates the body from the crown. The head and crown of the *classical crown decomposition* corresponds to a $(1, 1)$ -CD, the q -*expansion* for $q \in \mathbb{N}$ to a $(q, 1)$ -CD and the *weighted crown decomposition* for $q \in \mathbb{N}$ to a (q, x) -CD. The last known structure captured by Definition 1 is the *balanced expansion* which corresponds for $x, y \in \mathbb{N}$ to a (A_1, A_2, y, f, x) DBE. The essential new structural property of a DBE compared to a balanced expansion are the varying demands for A .

Let $G = (A \cup B, E)$ be a graph. For $A' \subseteq A$ we define $\mathcal{B}_{A'}$ as the components $Q \in \mathbb{CC}(G[B])$ with $N(Q) \subseteq A'$. The following theorem, easily derived as modification of the results of [5], gives the runtime to find a DBE and the existence guarantee of A_1 depending on the size or weight in the graph.

► **Theorem 2 (Demanded balanced expansion).** *Let $G = (A \cup B, E, w)$ be a graph with no isolated components in $\mathbb{CC}(G[B])$, i.e. every component of $\mathbb{CC}(G[B])$ contains at least one neighbor of A . Let $y \geq \max_{Q \in \mathbb{CC}(G[B])} w(Q)$ and $D = \{d_a\}_{a \in A}$ demands with $d_a \in \mathbb{N}$ for each $a \in A$. A demanded balanced expansion (A_1, A_2, y, f, D) can be computed in $\mathcal{O}(|V| |E|)$ time. Furthermore, if there is an $A' \subseteq A$ with $w(A') + w(V(\mathcal{B}_{A'})) \geq \sum_{a \in A'} d_a$, then $A_1 \neq \emptyset$.*

The next graph structure that we use for our kernelization algorithms is introduced by Casel et al. [5] and is a combination of a balanced connected partition and a weighted crown decomposition, which is called *balanced crown decomposition*. Formally it is defined as follows (see also Figure 1 (right) for an illustration).

► **Definition 3** (λ -balanced crown decomposition). *A λ -balanced crown decomposition of a graph $G = (V, E, w)$ is a tuple (C, H, \mathfrak{R}, f) , where $\{H, C, R\}$ is a partition of V , the set \mathfrak{R} is a partition of R , and $f: \mathbb{CC}(G[C]) \rightarrow H$, such that:*

1. $w(Q) \leq \lambda$ for each $Q \in \mathbb{CC}(G[C])$,
2. $f(Q) \in N(Q)$ for each $Q \in \mathbb{CC}(G[C])$,
3. $N(C) \subseteq H$,
4. $w(h) + w(f^{-1}(h)) > \lambda$ for each $h \in H$ and
5. $G[R']$ is connected and $\lambda < w(R') \leq 3\lambda$ for each $R' \in \mathfrak{R}$.

(λ, λ) -CD

We use λ -BCD as an abbreviation for a λ -balanced crown decomposition. Looking at the original definition of a λ -BCD in [5] (Definition 6), we shift the λ value by one, which allows us to change several inequalities between strict and non-strict for a clearer representation, while still keeping the definition the same. Furthermore, λ must be at least two in the original definition, but to simplify the understanding of the application to the problems considered in this paper, it makes more sense that a λ -BCD can also exist with $\lambda = 1$.

The authors in [5] provide an algorithm that finds a λ -BCD in polynomial time, as given in the following.

► **Theorem 4** (Balanced crown decomposition theorem, [5] Theorem 7). *Let $G = (V, E, w)$ be a graph and $\lambda \in \mathbb{N}$, such that each connected component in G has weight larger than λ . A λ -balanced crown decomposition (C, H, \mathfrak{R}, f) of G can be computed in $\mathcal{O}(r^2 |V| |E|)$ time, where $r = |H| + |\mathfrak{R}| < |V|$ is at most the size of a maximum $(\lambda + 1)$ -packing.*

We end the preliminary section with a formal definition of the subgraph packing problem. Given a graph $G = (V, E)$ and two parameters $r, \lambda \in \mathbb{N}$. We say that $P_1, \dots, P_m \subseteq V$ is a λ -packing if for all $i, j \in [m]$ with $i \neq j$ the induced subgraph $G[P_i]$ is connected, $|P_i| \geq \lambda$, and $P_i \cap P_j = \emptyset$. The task is to find a λ -packing of size at least r .

3 Improved Kernels for VI, wVI and wCOC

The vertex integrity of a graph models finding an optimal balance between removing vertices and keeping small connected parts of a graph. As a reminder: In the formal definition, a graph G and a parameter p are given. The task is to find a vertex set $S \subseteq V$ such that $|S| + \max_{Q \in \mathbb{CC}(G-S)} |Q|$ is at most p . In the weighted vertex integrity problem (wVI), i.e. the vertices have weights, the aim is that $w(S) + \max_{Q \in \mathbb{CC}(G-S)} w(Q)$ is at most p . We improve the vertex kernel of p^3 provided by Drange et al. [10] for both variants VI and wVI. To obtain such a kernel, they essentially established that vertices $v \in V$ with $w(v) + w(N(v)) > p$ belong to a solution if we have a yes-instance in hand. A simple counting argument after removing those vertices provides a p^3 vertex kernel. We improve this by employing crown decompositions, which offer valuable insights into the structural properties of vertex sets rather than focusing solely on individual vertices. Directly applying this method to problems like COC, as seen in prior works such as [5, 7, 19, 25], is challenging because we lack prior knowledge about the size of remaining components after solution removal. However, if we had at least a lower bound, we could theoretically safely reduce our instance accordingly. To establish such a bound, we engage in an interplay between packings and separators, where

the balanced crown decomposition (BCD) proves instrumental. By identifying reducible structures in the input instance through embedding a DBE into BCD after determining a suitable bound, we prove the following theorems regarding VI and wVI.

► **Theorem 5.** *The vertex integrity problem admits a vertex kernel of size $3p^2$ in time $\mathcal{O}(\log(p)|V|^4|E|)$.*

► **Theorem 6.** *The weighted vertex integrity problem admits a vertex kernel of size $3(p^2 + p^{1.5}p_\ell)$ in time $\mathcal{O}(\log(p)|V|^4|E|)$, where p_ℓ is at most the size of the largest component after removing a solution.*

Closely related to wVI is the weighted component order connectivity problem (wCOC). Given a vertex-weighted graph $G = (V, E, w)$ and two parameters $k, W \in \mathbb{N}$, the task is to find a vertex set $S \subseteq V$ such that $w(S) \leq k$ where each component weighs at most W . The techniques employed to derive the kernel for wVI can be seamlessly applied to wCOC, thereby enhancing the current state of the art vertex kernel of $kW(k + W) + k$. This kernel is also provided by Drange et al. [10] in a similar way as for wVI.

► **Theorem 7.** *The weighted component order connectivity problem admits a vertex kernel of size $3\mu(k + \sqrt{\mu}W)$, where $\mu = \max(k, W)$. Furthermore, such a kernel can be computed in time $\mathcal{O}(r^2k|V||E|)$, where r is the size of a maximum $(W + 1)$ -packing.*

Before we prove these theorems, we give some notations that we use in this section. We define them for the weighted case, as the unweighted case can be viewed in the same way with unit weights. We say that S is a solution if it satisfies $w(S) + \max_{Q \in \text{CC}(G-S)} w(Q) \leq p$ for wIV or $w(S) \leq k$ and $\max_{Q \in \text{CC}(G-S)} w(Q) \leq W$ for wCOC. We denote an instance of wIV by (G, p) and an instance of wCOC by (G, k, W) . We say that (G, p) or (G, k, W) is a *yes-instance* if there is a solution, otherwise, we say that it is a *no-instance*. Let $\mathcal{S} \subseteq 2^{V(G)}$ be all solutions for (G, p) . We define $p_\ell := \min_{S \in \mathcal{S}} (\max_{Q \in \text{CC}(G-S)} w(Q))$ which is the optimum lower bound on the size of the connected components after the removal of any solution; where for no-instances we set $p_\ell = p$. For all instances (G, p) of wVI, we assume $w(v) < p$ for every $v \in V$ and that G contains a connected component of weight more than p .

3.1 Vertex Integrity

The rest of this section is dedicated to the proof of Theorem 5. First, we discuss the reducible structure we are looking for and then show how to find it. Note that for $a, b \in \mathbb{N}$ with $a < b$ a (p, a) -CD is also a (p, b) -CD, but not vice versa. This means that even if we do not know p_ℓ exactly, any (p, c) -CD with $c \leq p_\ell$ is a (p, p_ℓ) -CD that can be used in the following lemma.

► **Lemma 8.**¹ *Let (G, p) be an instance of VI and let (C, H, f) be a (p, p_ℓ) -CD in G , such that $N(C) \subseteq H$. Then, (G, p) is a yes-instance if and only if $(G - \{H \cup C\}, p - |H|)$ is a yes-instance.*

In order to use Lemma 8, we must first determine a suitable lower bound for p_ℓ . Additionally, we need to ensure the existence of a corresponding weighted crown decomposition when the input graph size exceeds $3p^2$, and that we can find it efficiently. This is where the BCD comes into play. For $\lambda \in \mathbb{N}$ a λ -BCD (C, H, \mathfrak{R}, f) in G can only be computed within the components of G that have a size (or weight) greater than λ . For a better readability,

¹ This result is an incorrect simplification to describe the idea. Please refer to the Full Version for the more technical correct reduction.

we will consistently assume that when computing a λ -BCD, we disregard small components, meaning that if (C, H, \mathfrak{R}, f) is a λ -BCD for G , then $C \cup H \cup V(\mathfrak{R})$ are only the vertices that are contained in a component of size (or weight) more than λ of G .

We specify the following lemma directly in the weighted version to also use it later. The lemma provides a lower bound for p_ℓ by a λ -BCD (C, H, \mathfrak{R}, f) , where we essentially use that a λ -BCD is also a $(\lambda + 1)$ -packing of size $|H| + |\mathfrak{R}|$. This results from the fact that each element $R \in \mathfrak{R}$ has the size (or weight) $\lambda + 1$ and is connected and that for each $h \in H$ the subgraph $G[\{h\} \cup V(f^{-1}(h))]$ is connected and has at least the size (or weight) $\lambda + 1$. Note that the vertex sets $\{\{h\} \cup V(f^{-1}(h))\}_{h \in H} \cup \mathfrak{R}$ are pairwise disjoint.

► **Lemma 9.** *Let (G, p) be an instance of wVI and for $\lambda \in [p]$ let (C, H, \mathfrak{R}, f) be a λ -BCD in G . If $|H| + |\mathfrak{R}| > p$, then $\lambda < p_\ell$ for the instance $(G[C \cup H \cup V(\mathfrak{R})], p)$.*

Note that the lower bound $\lambda + 1$ of p_ℓ in Lemma 9 is applicable to the induced graph of $V' = C \cup H \cup V(\mathfrak{R})$ within G . This implies that the isolated components of $G - V'$, defined as having a size (or weight) of at most λ , can be safely removed. Their removal does not affect the decision problem of VI or wVI concerning p . Moreover, note that an additional advantage of a $(\lambda + 1)$ -BCD (C, H, \mathfrak{R}, f) is that the balanced part \mathfrak{R} can be upper bounded by $3(\lambda + 1)|\mathfrak{R}| \leq 3p_\ell|\mathfrak{R}| < 3p|\mathfrak{R}|$, while a suitable λ choice also upper bounds $|\mathfrak{R}|$ by p . In particular, if $H = C = \emptyset$ and $|\mathfrak{R}| \leq p$, then we would already have an instance with a size (or weight) of at most $3p^2$.

Clearly, a yes-instance cannot have a lower bound for p_ℓ larger than p as stated in the following corollary.

► **Corollary 10.** *Let (G, p) be an instance of wVI. If for a p -BCD (C, H, \mathfrak{R}, f) it holds that $|H| + |\mathfrak{R}| > p$, then (G, p) is a no-instance.*

The next lemma shows under which conditions we can find a (p, λ) -CD in G with respect to a λ -BCD and the current graph size.

► **Lemma 11.** *Let (G, p) be an instance of VI and for $\lambda \in [p]$ let (C, H, \mathfrak{R}, f) be a λ -BCD in G with $|H| + |\mathfrak{R}| \leq p$. If $|C| + |H| + |V(\mathfrak{R})| \geq 3p^2$, then $G' = G[C \cup H]$ contains a (p, λ) -CD in G . Furthermore, we can extract it from G' in time $\mathcal{O}(|V(G')||E(G')|) \subseteq \mathcal{O}(|V(G)||E(G)|)$.*

If we now combine Lemma 9 and Lemma 11, we can finally use Lemma 8 by finding a $\lambda \in [p - 1]$ such that a $(\lambda + 1)$ -BCD (C, H, \mathfrak{R}, f) satisfies $|H| + |\mathfrak{R}| \leq p$, while a λ -BCD $(C', H', \mathfrak{R}', f')$ satisfies $|H'| + |\mathfrak{R}'| > p$, which only adds a factor of $\mathcal{O}(\log(p))$ to the computation cost. Formulated more precisely: For $Q \in \mathbb{CC}(C)$ we have $|Q| \leq \lambda + 1 \leq p_\ell$ as $\lambda < p_\ell$ by Lemma 9. Further, we remove all vertices that are in components of size at most λ , which is a safe reduction rule as explained above. If the vertex size of the remaining input graph, i.e. $G[C \cup H \cup V(\mathfrak{R})]$ is at least $3p^2$, then we can find a (p, λ) -CD for G in $G[C \cup H]$ in polynomial time by Lemma 11, which is a reducible structure by Lemma 8.

With these facts in hand we can design a kernelization algorithm. It takes as input an instance (G, p) of VI and returns an equivalent instance with at most $3p^2$ vertices.

Find reducible structures (AlgVI)

1. Compute a p -BCD $(C_1, H_1, \mathfrak{R}_1, f_1)$. If $|H_1| + |\mathfrak{R}_1| > p$ return a trivial no-instance.
2. Compute a 1-BCD $(C_2, H_2, \mathfrak{R}_2, f_2)$. Let $V' = V(G) \setminus (C_2 \cup H_2 \cup V(\mathfrak{R}_2))$. If $|H_2| + |\mathfrak{R}_2| \leq p$ and $V' \neq \emptyset$, then return $(G - V', p)$.² If $|H_2| + |\mathfrak{R}_2| \leq p$ and $V' = \emptyset$, then compute a $(p, 1)$ -CD (H'_2, C'_2, f'_2) in $G[H_2 \cup C_2]$ and return $(G - \{H'_2 \cup C'_2\}, p - |H'_2|)$.

² Note that in this case V' is the set of isolated vertices in G , as we compute BCD only on components of size larger than $\lambda = 1$.

3. Otherwise, for $\lambda \in [p - 1]$ find a $(\lambda + 1)$ -BCD (C, H, \mathfrak{R}, f) and a λ -BCD $(C', H', \mathfrak{R}', f')$, such that $|H| + |\mathfrak{R}| \leq p$ and $|H'| + |\mathfrak{R}'| > p$. Let $V'' = V(G) \setminus (C \cup H \cup V(\mathfrak{R}))$. If $V'' \neq \emptyset$, then return $(G - V'', p)$. Otherwise, compute a $(p, \lambda + 1)$ -CD (H'', C'', f'') in $G[C \cup H]$ for G and return $(G - \{C'' \cup H''\}, p - |H''|)$.

We already explained why we can safely remove V' or V'' in steps 2 and 3 if these are not empty. Note that if V' or V'' is empty, then the vertex sets of the corresponding BCD's, i.e. $C_2, H_2, V(\mathfrak{R}_2)$ and $C, H, V(\mathfrak{R})$, respectively, form a partition of $V(G)$. The correctness of step 1 is implied by Corollary 10. Step 2 is correct because $p_\ell \geq 1$ and if $V' = \emptyset$ we obtain by Lemma 11 and $|V| = |C_2| + |H_2| + |V(\mathfrak{R}_2)| \geq 3p^2$ that there must be a $(p, 1)$ -CD if $|H_2| + |\mathfrak{R}_2| \leq p$. Analogously, for step 3 as $p_\ell \geq \lambda + 1$ by Lemma 9 in this step. For the overall correctness of algorithm ALGVI, it remains to be proven that if we reach step 3, the required λ exists. For a λ -BCD in a binary search, we cannot ensure that $|H| + |\mathfrak{R}|$ increases with decreasing λ values, because the sizes of H and \mathfrak{R} are not necessarily monotonic with respect to λ . Note, for example, that the elements in \mathfrak{R} have a size range from $\lambda + 1$ to 3λ . For a successful binary search, however, it is sufficient to know that there is a λ that satisfies the desired properties for step 3. Since we only enter this step if the extreme cases (p -BCD and 1-BCD) in step 1 and 2 hold, there has to exist at least one such λ value in-between.

Finally, we prove the specified running time of Theorem 5, which completes the proof of Theorem 5. Note that for a yes-instance we have to call the algorithm ALGVI at most $|V|$ times to guarantee the desired kernel.

► **Lemma 12.** *Algorithm ALGVI runs in time $\mathcal{O}(\log(p)|V|^3|E|)$.*

3.2 Weighted Vertex Integrity and Component Order Connectivity

In this section, we shift our focus to the weighted variants, namely wVI and wCOC. While utilizing the packing size of the associated BCD offers a starting point for deriving a lower bound for VI, it proves insufficient for improvements in the weighted setting. Therefore, we integrate the weight of the separator H within a BCD (C, H, \mathfrak{R}, f) into our analysis. Additionally, we incorporate two distinct DBE's into a BCD in case a reduction is not achievable within the respective setting. This approach enables us to establish a tighter lower bound for p_ℓ , estimate the remaining instance size more accurately to obtain the desired kernelization results (cf. Theorems 6 and 7), or identify a no-instance.

Let us start by explaining the type of reducible structure we are searching for.

► **Lemma 13.**³ *Let $G = (V, E, w)$ be a vertex weighted graph, $a, b \in \mathbb{N}$, (H, H', b, f, D) a DBE in G with $D = \{d_h\}_{h \in H}$, where $d_h = a - 2 + b \cdot (w(h) + 1)$ for each $h \in H$, and let $C = f^{-1}(H)$ with $N(C) \subseteq H$.*

1. Let (G, p) be an instance of the weighted vertex integrity problem, $a = p$ and $1 \leq b \leq p_\ell$. Then, (G, p) is a yes-instance if and only if $(G - \{H \cup C\}, p - w(H))$ is a yes-instance.
2. Let (G, k, W) be an instance of the weighted component order connectivity problem and $a, b = W$. Then, (G, k, W) is a yes-instance if and only if $(G - \{H \cup C\}, k - w(H), W)$ is a yes-instance.

We are now introducing the two DBE's mentioned in conjunction with a BCD. Let $G = (V, E, w)$ be a vertex weighted graph, $a, s, \lambda \in \mathbb{N}$, (C, H, \mathfrak{R}, f) a λ -BCD with $\lambda \in [a]$ and $|H| + |\mathfrak{R}| \leq s$, where $\max_{v \in V} w(v) \leq \max(a, s) =: \mu$. Let $D^Y = \{d_h^Y\}_{h \in H}$ and

³ This result is an incorrect simplification to describe the idea. Please refer to the Full Version for the more technical correct reduction.

$D^Z = \{d_h^Z\}_{h \in H}$ be demands, where $d_h^Y = a - 2 + \lambda \cdot (w(h) + 1)$ and $d_h^Z = w(h) - 1 + (\sqrt{s} + 1)\lambda$ for each $h \in H$. Let $(Y_1, Y_2, \lambda, f_Y, D^Y)$ and $(Z_1, Z_2, \lambda, f_Z, D^Z)$ be DBE's in $G[C \cup H]$ with $Y_1, Y_2, Z_1, Z_2 \subseteq H$ and $f_Y^{-1}(Y_1), f_Y^{-1}(Y_2), f_Z^{-1}(Z_1), f_Z^{-1}(Z_2) \subseteq \mathbb{CC}(C)$. Observe that if $Y_1 \neq \emptyset$ (resp. $Z_1 \neq \emptyset$) then this set separates $f_Y^{-1}(Y_1)$ (res. $f_Z^{-1}(Z_1)$) from the rest of the graph, since H separates C from the rest of the graph. Thus, if $a = p$ and $\lambda \leq p_\ell$ considering wVI or $a, \lambda = W$ considering wCOC and $Y_1 \neq \emptyset$, then in both cases we would have a reducible structure for the corresponding problem (cf. Lemma 13). We conclude this section by presenting two crucial lemmas. These lemmas serve as the foundation for designing the algorithms needed to prove Theorems 6 and 7. The first lemma aims to estimate the instance size, while the second lemma helps establish a more precise lower bound for p_ℓ or identify instances with no feasible solutions.

► **Lemma 14.** *Let $w(Z_1) \leq 2\mu^{1.5}$. If $Y_1 = \emptyset$, then $w(V) < 3\mu(s + \sqrt{\mu}\lambda)$.*

► **Lemma 15.** *If $w(Z_1) > 2\mu^{1.5}$, then there is no separator S such that $w(S) \leq s$ and $\max_{Q \in \mathbb{CC}(G-S)} w(Q) \leq \lambda$.*

4 Kernels for Component Order Connectivity

In this section, we consider COC, a refined version of VI. Given a graph $G = (V, E)$ and parameters $k, W \in \mathbb{N}$, COC aims to remove at most k vertices so that resulting components have at most W vertices each. We present a kernelization algorithm that provides a $2kW$ vertex-kernel in an FPT-runtime when parameterized by the size of a maximum $(W + 1)$ -packing, as stated in the following theorem.

► **Theorem 16.** *A vertex kernel of size $2kW$ for the component order connectivity problem can be computed in time $\mathcal{O}(r^3|V||E| \cdot r^{\min(3r, k)})$, where $r \leq k$ is the size of a maximum $(W + 1)$ -packing.*

A kernel of size $2kW$ is also presented in [19], however with an FPT-runtime in the parameter W and using linear programming methods. In contrast, our result has an FPT-runtime in the parameter of a maximum $(W + 1)$ -packing $r \leq k$ and is fully combinatorial. Moreover, we showcase how our algorithm transforms into a polynomial one for two cases: when $W = 1$ (Vertex Cover) and for claw-free graphs. While generalizing our FPT-runtime algorithm into polynomial time eludes us, our understanding of crown decomposition's structural properties holds promise for future progress.

For VI, wVI, and wCOC, crown structures were integrated into the head and crown of the BCD. In contrast, we now incorporate them into the body of the BCD. Of particular interest are the so-called strictly reducible pairs introduced by Kumar and Lokshtanov [19] who prove that such a structure must exist in graphs with more than $2kW$ vertices.

► **Definition 17** ((strictly) reducible pair). *For a graph $G = (V, E)$ and a parameter $W \in \mathbb{N}$, a pair (A, B) of vertex disjoint subsets of V is a reducible pair for COC if the following conditions are satisfied:*

- $N(B) \subseteq A$.
- The size of each $Q \in \mathbb{CC}(G[B])$ is at most W .
- There is an assignment function $g: \mathbb{CC}(G[B]) \times A \rightarrow \mathbb{N}_0$, such that
 - for all $Q \in \mathbb{CC}(G[B])$ and $a \in A$, if $g(Q, a) \neq 0$, then $a \in N(Q)$
 - for all $a \in A$ we have $\sum_{Q \in \mathbb{CC}(G[B])} g(Q, a) \geq 2W - 1$,
 - for all $Q \in \mathbb{CC}(G[B])$ we have $\sum_{a \in A} g(Q, a) \leq |Q|$.

In addition, if there exists an $a \in A$ such that $\sum_{Q \in \text{CC}(G[B])} g(Q, a) \geq 2W$, then (A, B) is a *strictly reducible pair*.

We say (A, B) is a minimal (strictly) reducible pair if there does not exist a (strictly) reducible pair (A', B') with $A' \subset A$ and $B' \subseteq B$. A (strictly) reducible pair is basically a weighted crown decomposition where A is the head and B is the crown.

In essence, the kernelization algorithm outlined below focuses on analyzing pairs (A, B) in a W -BCD. We show that the head vertices A have specific traits in a W -BCD, making it possible to locate them. Structurally, the algorithm operates as a bounded search tree, hence it is presented recursively. It takes an instance (G, k, W) of COC as input, where $|V(G)| > 2kW$ and each component of G has at least $W + 1$ vertices. The size limits for the input graph are not arbitrary; if $|V(G)| \leq 2kW$, there is nothing to do, and removing components smaller than $W + 1$ is a safe reduction.

For $V' \subseteq V$ we define $\text{sep}_W(V') \in \mathbb{N}$ as the cardinality of a minimum W -separator in $G[V']$, and $\text{arg sep}_W(V') \subset V$ as an argument suitable to this cardinality. For a graph $G' \subseteq G$ let $G^{>W}(G')$ be the graph obtained by removing all components of size at most W from G . Lastly, for a W -BCD (C, H, \mathfrak{R}, f) we define $\mathfrak{R}' := \{R \in \mathfrak{R} \mid |R| > 2W \text{ and } \text{sep}_W(R) = 1\}$ and $S_{\mathfrak{R}'} := \bigcup_{R \in \mathfrak{R}'} \text{arg sep}_W(R)$. (uniqueness of $\text{arg sep}_W(R)$ for $R \in \mathfrak{R}'$ is shown in the full version [6]).

Find reducible structures (AlgCOC)

1. Compute a W -BCD (C, H, \mathfrak{R}, f) in G and initialize $t = |H| + |\mathfrak{R}|$ and $S = \emptyset$.
2. Let $G' = G^{>W}(G - S)$. If G' is an empty graph return a trivial yes-instance. Otherwise, compute a W -BCD (C, H, \mathfrak{R}, f) in G' .
 - a. If $|H| + |\mathfrak{R}| > k$ return a trivial no-instance.
 - b. Let \mathcal{Q} be the connected components of size at most W in $G - S$. Let $A = S \cup H$ and $B = C \cup V(\mathcal{Q})$. Compute a DBE $(A_1, A_2, W, f, 2W - 1)$ in $G[A \cup B]$ by using Theorem 2. If $A_1 \neq \emptyset$, then terminate the algorithm and return $(G - (A_1 \cup f^{-1}(A_1)), k - |A_1|, W)$.
3. If the depth of the recursion is more than $\min(3t, k)$, then break.
4. For each $v \in H \cup S_{\mathfrak{R}'}$:
 - Add v to S and recurse from step 2.
5. Return a trivial no-instance.

The interesting part of the algorithm is the localization of a minimal strictly reducible pair if it exists in the graph. Let (A, B) be a minimal strictly reducible pair in G . As already mentioned, algorithm ALGCOC is basically a bounded search tree with a working vertex set S , which are potential head vertices. The crucial step is ensuring that S consistently matches the vertex set A . Once this alignment is achieved, localization of (A, B) is assured, as step 2b (specifically Theorem 2) guarantees its extraction.

We conclude this part of the section by presenting a crucial lemma along with its implication for a depth-wise progression within the bounded search tree towards $S = A$ in algorithm ALGCOC. Let (A, B) be a minimal strictly reducible pair in $G = (V, E)$ and let $S \subset V$. Let (C, H, \mathfrak{R}, f) be a W -BCD in $G^{>W}(G - S)$.

► **Lemma 18.** *If $S \cap B = \emptyset$ and $S \not\subseteq A$, then $(H \cup S_{\mathfrak{R}'}) \cap A \neq \emptyset$.*

If we reach step 4, we have no reducible pair in hand so far and can therefore assume that $S \neq A$. Thus Lemma 18 provides the following relation to the algorithm: if $S \subset A$ (where S is possibly empty), then we can find at least one vertex of A in $H \cup S_{\mathfrak{R}'}$. Since the algorithm considers expanding S for each vertex of $H \cup S_{\mathfrak{R}'}$, at least one of which comes from A , and repeats this process with the resulting vertex set, we finally arrive at the case $S = A$.

Next, we introduce another algorithm designed to compute a $2kW$ vertex kernel for COC. This algorithm shares the same principles as ALGCOC but exhibits polynomial running times for two specific cases.

Find reducible structures (AlgCOC-2)

1. Initialize $S = \emptyset$ and $G' = G$.
2. While $G' \neq (\emptyset, \emptyset)$:
 - a. Compute a W -BCD (C, H, \mathfrak{R}, f) in G' , such that for a minimal strictly reducible pair (A, B) in G we have $B \cap S_{\mathfrak{R}'} = \emptyset$.
 - b. If $|H| + |\mathfrak{R}| > k$, or $\mathfrak{R}' = \emptyset$ and $H = \emptyset$, then terminate the while-loop.
 - c. Add the vertices $S_{\mathfrak{R}'}$ and H to S .
 - d. Let \mathcal{Q} be the connected components of size at most W in $G - S$. Let $A = S \cup H$, $B = C \cup V(\mathcal{Q})$. Compute a DBE $(A_1, A_2, W, f, 2W - 1)$ in $G[A \cup B]$ by using Theorem 2. If $A_1 \neq \emptyset$, then terminate the algorithm and return $(G - (A_1 \cup f^{-1}(A_1)), k - |A_1|, W)$.
 - e. Update G' by $G'^{>W}(G - S)$.
3. Return a trivial no-instance.

The correctness proof of ALGCOC-2 can be found in the extended version [6]. The crucial difference to the previous algorithm is step 2a. Unfortunately, we have not succeeded in finding a polynomial algorithm in general for this step.

To introduce a novel approach for computing a $2k$ vertex kernel for the vertex cover problem, we present a result that establishes a $2kW$ vertex kernel for COC, applicable for any given W . However, the corresponding algorithm achieves polynomial runtime only when $W = 1$. This limitation arises because computing a maximum $(W + 1)$ -packing is polynomially solvable only for $W = 1$ where it corresponds to a maximum matching. To understand how we can apply algorithm ALGCOC-2 (basically ensuring $B \cap S_{\mathfrak{R}'} = \emptyset$ in any iteration of the while-loop) we need to extend the algorithm to compute a W -BCD (C, H, \mathfrak{R}, f) . First observe that $\mathcal{P} = \mathfrak{R} \cup \{\{h\} \cup V(f^{-1}(h))\}_{h \in H}$ forms a $(W + 1)$ -packing of size $|H| + |\mathfrak{R}|$. In order to find \mathfrak{R} and H in the algorithm computing a W -BCD there are two according working sets, \mathfrak{R}' and H' , that always guarantee a $(W + 1)$ -packing of size $|H'| + |\mathfrak{R}'|$. In particular, these sets measure the progress in a sense that the corresponding packing can only increase. The crucial point now is that we can start the algorithm with an arbitrary maximal $(W + 1)$ -packing and if this is a maximum $(W + 1)$ -packing, then after the termination of the algorithm, \mathcal{P} corresponds also to a maximum $(W + 1)$ -packing. In fact, the original algorithm initializes \mathfrak{R}' in the beginning as the connected components of the input graph and $H' = \emptyset$, but it is also possible to initialize \mathfrak{R}' as a maximal $(W + 1)$ -packing instead (see [5], proof sketch of Theorem 7). The relationship between a W -BCD, where \mathcal{P} represents a maximum $(W + 1)$ -packing, and a reducible pair (A, B) yields a significant property: every element in \mathcal{P} intersects with at most one vertex of A . However, if a vertex of B is in $S_{\mathfrak{R}'}$ in algorithm ALGCOC-2, then the according $R \in \mathfrak{R}'$ must contain at least two vertices of A . This contradicts the aforementioned property. These findings form the key point of the following theorem.

► **Theorem 19.** *For $W = 1$, i.e. for the vertex cover problem, algorithm ALGCOC-2 works correctly and provides a $2k$ vertex kernel in polynomial time.*

We also demonstrate the application of ALGCOC-2 for claw-free graphs, i.e., graphs without induced $K_{1,3}$'s. To achieve this, we use a vertex partitioning algorithm tailored for such graphs, as proposed by Borndörfer et al. [4]. Essentially, we establish that claw-free

graphs can be represented by a W -BCD (C, H, \mathfrak{R}, f) with C and H being empty, and at most one element of \mathfrak{R} exceeding size $2W$. Utilizing this along with Lemma 18 ensures that $B \cap S_{\mathfrak{R}'} = \emptyset$ in any iteration of the while-loop and provides basically the following theorem.

► **Theorem 20.** *The component order connectivity problem admits a $2kW$ vertex-kernel on claw-free graphs in polynomial time.*

References

- 1 Kunwarjit S. Bagga, Lowell W. Beineke, Wayne Goddard, Marc J. Lipman, and Raymond E. Pippert. A survey of integrity. *Discret. Appl. Math.*, 37/38:13–28, 1992. doi:10.1016/0166-218X(92)90122-Q.
- 2 Curtis A Barefoot, Roger Entringer, and Henda Swart. Vulnerability in graphs—a comparative survey. *J. Combin. Math. Combin. Comput.*, 1(38):13–22, 1987.
- 3 Matthias Bentert, Klaus Heeger, and Tomohiro Koana. Fully polynomial-time algorithms parameterized by vertex integrity using fast matrix multiplication. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms, ESA 2023, September 4–6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPICs*, pages 16:1–16:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.ESA.2023.16.
- 4 Ralf Borndörfer, Katrin Casel, Davis Issac, Aikaterini Niklanovits, Stephan Schwartz, and Ziena Zeif. Connected k -partition of k -connected graphs and c -claw-free graphs. In Mary Wootters and Laura Sanità, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2021, August 16–18, 2021, University of Washington, Seattle, Washington, USA (Virtual Conference)*, volume 207 of *LIPICs*, pages 27:1–27:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.APPROX/RANDOM.2021.27.
- 5 Katrin Casel, Tobias Friedrich, Davis Issac, Aikaterini Niklanovits, and Ziena Zeif. Balanced crown decomposition for connectivity constraints. In *29th Annual European Symposium on Algorithms, ESA 2021, September 6–8, 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPICs*, pages 26:1–26:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ESA.2021.26.
- 6 Katrin Casel, Tobias Friedrich, Aikaterini Niklanovits, Kirill Simonov, and Ziena Zeif. Combining crown structures for vulnerability measures. *CoRR*, abs/2405.02378, 2024. doi:10.48550/arXiv.2405.02378.
- 7 Jianer Chen, Henning Fernau, Peter Shaw, Jianxin Wang, and Zhibiao Yang. Kernels for packing and covering problems. *Theor. Comput. Sci.*, 790:152–166, 2019. doi:10.1016/J.TCS.2019.04.018.
- 8 Lane H Clark, Roger C Entringer, and Michael R Fellows. Computational complexity of integrity. *J. Combin. Math. Combin. Comput.*, 2:179–191, 1987.
- 9 Rodney G Downey and Michael Ralph Fellows. *Parameterized complexity*. Springer Science & Business Media, 2012. doi:10.1007/978-1-4612-0515-9.
- 10 Pål Grønås Drange, Markus S. Dregi, and Pim van ’t Hof. On the computational complexity of vertex integrity and component order connectivity. *Algorithmica*, 76(4):1181–1202, 2016. doi:10.1007/s00453-016-0127-x.
- 11 Michael R Fellows and Sam Stueckle. The immersion order, forbidden subgraphs and the complexity of network integrity. *J. Combin. Math. Combin. Comput.*, 6(1):23–32, 1989.
- 12 Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: theory of parameterized preprocessing*. Cambridge University Press, 2019. doi:10.1017/9781107415157.
- 13 Tatsuya Gima, Tesshu Hanaka, Masashi Kiyomi, Yasuaki Kobayashi, and Yota Otachi. Exploring the gap between treedepth and vertex cover through vertex integrity. *Theor. Comput. Sci.*, 918:60–76, 2022. doi:10.1016/J.TCS.2022.03.021.

- 14 Tatsuya Gima, Tesshu Hanaka, Yasuaki Kobayashi, Ryota Murai, Hirota Ono, and Yota Otachi. Structural parameterizations of vertex integrity. In Ryuhei Uehara, Katsuhisa Yamanaka, and Hsu-Chun Yen, editors, *WALCOM: Algorithms and Computation - 18th International Conference and Workshops on Algorithms and Computation, WALCOM 2024, Kanazawa, Japan, March 18-20, 2024, Proceedings*, volume 14549 of *Lecture Notes in Computer Science*, pages 406–420. Springer, 2024. doi:10.1007/978-981-97-0566-5_29.
- 15 Tatsuya Gima and Yota Otachi. Extended MSO model checking via small vertex integrity. *Algorithmica*, 86(1):147–170, 2024. doi:10.1007/S00453-023-01161-9.
- 16 Tesshu Hanaka, Michael Lampis, Manolis Vasilakis, and Kanae Yoshiwatari. Parameterized vertex integrity revisited. *CoRR*, abs/2402.09971, 2024. doi:10.48550/arXiv.2402.09971.
- 17 Ashwin Jacob, Diptapriyo Majumdar, and Venkatesh Raman. Expansion lemma - variations and applications to polynomial-time preprocessing. *Algorithms*, 16(3):144, 2023. doi:10.3390/A16030144.
- 18 Dieter Kratsch, Ton Kloks, and Haiko Müller. Measuring the vulnerability for classes of intersection graphs. *Discret. Appl. Math.*, 77(3):259–270, 1997. doi:10.1016/S0166-218X(96)00133-3.
- 19 Mithilesh Kumar and Daniel Lokshtanov. A $2k$ kernel for l -component order connectivity. In Jiong Guo and Danny Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*, volume 63 of *LIPICs*, pages 20:1–20:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.IPEC.2016.20.
- 20 Michael Lampis and Valia Mitsou. Fine-grained meta-theorems for vertex integrity. In Hee-Kap Ahn and Kunihiko Sadakane, editors, *32nd International Symposium on Algorithms and Computation, ISAAC 2021, December 6-8, 2021, Fukuoka, Japan*, volume 212 of *LIPICs*, pages 34:1–34:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ISAAC.2021.34.
- 21 Euiwoong Lee. Partitioning a graph into small pieces with applications to path transversal. *Math. Program.*, 177(1-2):1–19, 2019. doi:10.1007/s10107-018-1255-7.
- 22 Wenjun Li and Binhai Zhu. A $2k$ -kernelization algorithm for vertex cover based on crown decomposition. *Theor. Comput. Sci.*, 739:80–85, 2018. doi:10.1016/J.TCS.2018.05.004.
- 23 Yinkui Li, Shenggui Zhang, and Qilong Zhang. Vulnerability parameters of split graphs. *Int. J. Comput. Math.*, 85(1):19–23, 2008. doi:10.1080/00207160701365721.
- 24 Mingyu Xiao. Linear kernels for separating a graph into components of bounded size. *J. Comput. Syst. Sci.*, 88:260–270, 2017. doi:10.1016/J.JCSS.2017.04.004.
- 25 Mingyu Xiao and Shaowei Kou. Kernelization and parameterized algorithms for 3-path vertex cover. In T. V. Gopal, Gerhard Jäger, and Silvia Steila, editors, *Theory and Applications of Models of Computation - 14th Annual Conference, TAMC 2017, Bern, Switzerland, April 20-22, 2017, Proceedings*, volume 10185 of *Lecture Notes in Computer Science*, pages 654–668, 2017. doi:10.1007/978-3-319-55911-7_47.

Linear-Time MaxCut in Multigraphs Parameterized Above the Poljak-Turzík Bound

Jonas Lill 

Department of Computer Science, ETH Zürich, Switzerland

Kalina Petrova¹  

Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Simon Weber  

Department of Computer Science, ETH Zürich, Switzerland

Abstract

MAXCUT is a classical NP-complete problem and a crucial building block in many combinatorial algorithms. The famous *Edwards-Erdős bound* states that any connected graph on n vertices with m edges contains a cut of size at least $\frac{m}{2} + \frac{n-1}{4}$. Crowston, Jones and Mnich [Algorithmica, 2015] showed that the MAXCUT problem on simple connected graphs admits an FPT algorithm, where the parameter k is the difference between the desired cut size c and the lower bound given by the Edwards-Erdős bound. This was later improved by Etscheid and Mnich [Algorithmica, 2017] to run in parameterized linear time, i.e., $f(k) \cdot O(m)$. We improve upon this result in two ways: Firstly, we extend the algorithm to work also for *multigraphs* (alternatively, graphs with positive integer weights). Secondly, we change the parameter; instead of the difference to the Edwards-Erdős bound, we use the difference to the *Poljak-Turzík bound*. The Poljak-Turzík bound states that any weighted graph G has a cut of size at least $\frac{w(G)}{2} + \frac{w_{MSF}(G)}{4}$, where $w(G)$ denotes the total weight of G , and $w_{MSF}(G)$ denotes the weight of its minimum spanning forest. In connected simple graphs the two bounds are equivalent, but for multigraphs the Poljak-Turzík bound can be larger and thus yield a smaller parameter k . Our algorithm also runs in parameterized linear time, i.e., $f(k) \cdot O(m+n)$.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Mathematics of computing → Combinatorial optimization; Theory of computation → Fixed parameter tractability

Keywords and phrases Fixed-parameter tractability, maximum cut, Edwards-Erdős bound, Poljak-Turzík bound, multigraphs, integer-weighted graphs

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.2

Funding *Kalina Petrova*: Swiss National Science Foundation, grant no. CRSII5 173721. This project has received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 101034413.

Simon Weber: Swiss National Science Foundation under project no. 204320.

1 Introduction

The MAXCUT(G, c) problem is the problem of deciding whether a given graph G contains a cut of size at least c . It has been known for a very long time that this problem is NP-complete, in fact it was one of Karp’s 21 NP-complete problems [9]. The MAXCUT problem has been intensely studied from various angles such as random graph theory and combinatorics, but also approximation and parameterized complexity. It has numerous applications in areas such as physics and circuit design; for more background on the MAXCUT problem we refer to the excellent survey [14].

¹ Parts of this research was conducted while Kalina Petrova was at the Department of Computer Science, ETH Zürich, Switzerland.



There are many lower bounds on the maximum cut size $\mu(G)$ of a given graph G . If G is a graph with m edges, a trivial lower bound is $\mu(G) \geq \frac{m}{2}$. This can be shown easily using the probabilistic method, as first done by Erdős [4]. Clearly, $\text{MAXCUT}(G, c)$ is thus easily solvable if $c \leq \frac{m}{2}$. But what if c is larger? At which point does the MAXCUT problem become difficult? It turns out that already $c = \frac{m}{2} + \epsilon m$ for any fixed $\epsilon > 0$ makes the problem NP-hard [7]. However, as long as the difference $c - \frac{m}{2}$ is just a constant, $\text{MAXCUT}(G, c)$ is still polynomial-time solvable: Mahajan and Raman showed in 1999 [11] that $\text{MAXCUT}(G, \frac{m}{2} + k)$ is fixed-parameter tractable (FPT), i.e., it can be solved in time $f(k) \cdot n^{O(1)}$. This started off the study of *parameterized algorithms above guaranteed lower bounds*.

By the time this FPT algorithm was found, $\frac{m}{2}$ was no longer the best-known lower bound for $\mu(G)$. Already more than 20 years earlier, Edwards showed the following lower bound that was previously conjectured by Erdős, and is thus now known as the Edwards-Erdős bound.

► **Theorem 1** (Edwards-Erdős bound [2, 3]). *For any connected simple graph G with n vertices and m edges, $\mu(G) \geq \frac{m}{2} + \frac{n-1}{4}$.*

Unlike the previous bound of $\frac{m}{2}$, this bound is tight for an infinite class of graphs, for example the odd cliques. It remained open for quite a while whether $\text{MAXCUT}(G, \frac{m}{2} + \frac{n-1}{4} + k)$ would also be fixed-parameter tractable, i.e., whether the parameter k could be reduced by $\frac{n-1}{4}$ compared to the previous result by Mahajan et al. This question was answered in the positive by Crowston, Jones and Mnich, who proved the following theorem.

► **Theorem 2** (Crowston, Jones, Mnich [1, Thm. 1]). *There is an algorithm that computes, for any connected graph G with n vertices and m edges and any integer k , in time $2^{O(k)} \cdot n^4$ a cut of G of size at least $\frac{m}{2} + \frac{n-1}{4} + k$, or decides that no such cut exists.*

This algorithm has later been improved to run in linear time (in terms of m) by Etscheid and Mnich [5]. However, this improvement only holds for deciding the existence of such a cut, and not for computing a cut if one exists.

We would like to highlight another classic lower bound on the size of the maximum cut of a graph, nicknamed the “spanning tree” bound: Any connected graph on n vertices has a cut of size at least $n - 1$, since it contains a spanning tree of this size and trees are bipartite. Note that this bound is incomparable to the Edwards-Erdős bound. In 2018, Madathil, Saurabh, and Zehavi [10] showed that $\text{MAXCUT}(G, n - 1 + k)$ is also fixed-parameter tractable.

In 1986, Poljak and Turzík improved upon the Edwards-Erdős bound by replacing the term $n - 1$ with the size of the minimum spanning tree (or forest in disconnected graphs), thus obtaining the following lower bound for maximum cuts in weighted graphs.

► **Theorem 3** (Poljak-Turzík bound [13]). *For any graph $G = (V, E)$ with weight function $w : E \rightarrow \mathbb{R}_{>0}$, we have $\mu(G) \geq \frac{w(G)}{2} + \frac{w_{MSF}(G)}{4}$, where $w(G) = \sum_{e \in E} w(e)$ and $w_{MSF}(G)$ denotes the weight of a minimum-weight spanning forest of G .*

It is easy to see that Theorem 3 implies the bound in Theorem 1 both for (unweighted) simple graphs and multigraphs. In unweighted simple graphs it is actually equivalent to Theorem 1, while on multigraphs and positive integer-weighted graphs it can be strictly larger.

The authors of Theorem 2 thus posed as their major open question whether their algorithm could be extended to solve $\text{MAXCUT}(G, \frac{m}{2} + \frac{n-1}{4} + k)$ on multigraphs as well. We answer this question in the positive, and improve the result further by replacing the Edwards-Erdős bound with the Poljak-Turzík bound.

1.1 Results

We provide a parameterized linear time algorithm for deciding MAXCUT in multigraphs and positive integer-weighted (simple) graphs above the Poljak-Turzík bound. A multigraph can be easily turned into a positive integer-weighted graph and vice versa; in the rest of this paper we phrase all of our results and proofs in terms of positive integer-weighted graphs for better legibility.

► **Theorem 4.** *There is an algorithm that decides for any graph $G = (V, E)$ with weight function $w : E \rightarrow \mathbb{N}$ and any integer k , in time $2^{O(k)} \cdot O(|E| + |V|)$, whether a cut of G of size at least $\frac{w(G)}{2} + \frac{w_{MSF}(G)}{4} + k$ exists.*

Using the same techniques we can also get a parameterized quadratic-time algorithm to compute such a cut, if one exists.

► **Theorem 5.** *There is an algorithm that computes for any graph $G = (V, E)$ with weight function $w : E \rightarrow \mathbb{N}$ and any integer k , in time $2^{O(k)} \cdot O(|E| \cdot |V|)$, a cut of G of size at least $\frac{w(G)}{2} + \frac{w_{MSF}(G)}{4} + k$, if one exists.*

We would like to point out that Theorem 4 is a strict improvement on the linear-time algorithm from [5] in two ways: Firstly we increase the types of graphs the algorithm is applicable to, and secondly we also strictly decrease the parameter for some instances. The following observation shows that this decrease of parameter can be significant.

► **Observation 6.** *There exist sequences of positive-integer-weighted graphs $(G_i)_{i \in \mathbb{N}}$ and integers $(c_i)_{i \in \mathbb{N}}$ such that Theorem 4 yields a polynomial-time algorithm to solve $\text{MAXCUT}(G_i, c_i)$, but when replacing $w_{MSF}(G)$ by $n - 1$, it does not.*

Proof. Let G_i be a tree on $i + 1$ vertices where each edge has weight 2. Then, the Poljak-Turzík bound yields $\mu(G_i) \geq \frac{2i}{2} + \frac{2i}{4} = \frac{6}{4}i$, while the Edwards-Erdős bound only yields $\mu(G_i) \geq \frac{2i}{2} + \frac{i+1-1}{4} = \frac{5}{4}i$. Thus, if we set $c_i = \frac{6}{4}i + k$ for some constant k , then Theorem 4 yields a $2^{O(k)} \cdot \text{poly}(i) = \text{poly}(i)$ algorithm for $\text{MAXCUT}(G_i, c_i)$, while with the Edwards-Erdős bound it would yield a $2^{O(k + \frac{1}{4}i)} \cdot \text{poly}(i)$ algorithm, which is not polynomial. ◀

1.2 Algorithm Overview

Our algorithm works in a very similar fashion to the one in [1]. We use a series of reduction rules that can reduce the input graph down to a graph with no edges. While performing this reduction, we either prove that G has a cut of the desired size, or we collect a set S of $O(k)$ vertices such that $G - S$ is a uniform-clique-forest, i.e., a graph in which every biconnected component is a clique in which every edge has the same weight. Given such a set S , we can then compute the maximum cut of G exactly: We iteratively test all possibilities of partitioning the vertices in S between the two sides of the cut, and then compute the maximum cut of G assuming that the vertices of S are indeed partitioned like this. To do this, we use a similar approach as in [1]: We compute the maximum cut of $G - S$ with *weighted vertices*. In this setting, each vertex v in $G - S$ specifies a weight $w_0(v)$ and $w_1(v)$ for both possible sides of the cut v may land in. The value of a cut is given by the total weight of the

cut edges plus the sum of the correct weight for each vertex. To use this problem to compute the maximum cut of G , we set the weights of each vertex v in $G - S$ according to the total weight of the edges between v and S that are cut in the assumed partition of S . Maximizing over all possible partitions for S gives the maximum cut of G .

While we use very similar techniques as in [1, 5], our main technical contribution lies in the reduction rules. Our reduction rules have to be more specific, i.e., each reduction rule has a stronger precondition. This is due to the fact that when performing any reduction, the change in the weight of a minimum spanning forest (as needed for the Poljak-Turzík bound) is much more difficult to track than the number of vertices in the graph (as needed for the Edwards-Erdős bound). Since our rules are more specific, we also need twice as many rules as in [1] (and one more rule than [5]) to ensure that always at least one rule is applicable to a given graph.

2 Preliminaries

In the rest of this paper we consider every graph to be a simple graph $G = (V, E)$, where V is the set of vertices, and $E \subseteq \binom{V}{2}$ is the set of edges. A graph is weighted if it is equipped with a positive integer edge-weight function $w : E \rightarrow \mathbb{N}$. For any two disjoint subsets $A, B \subseteq V$ we denote by $E(A, B)$ the set of edges between A and B , by $w(A, B)$ the total weight of the edges in $E(A, B)$, and by $\min(A, B)$ the minimum weight of any edge in $E(A, B)$. For a subset $A \subseteq V$, we denote by $N(A)$ the set of vertices in $V \setminus A$ that have a neighbor in A .

A *cut* is a subset $C \subseteq V$, and the *weight* of a cut C is the total weight of the edges connecting a vertex in C to a vertex in $V \setminus C$, i.e., $w(C) = w(C, V \setminus C)$.

For any set $A \subseteq V$ we write $G[A]$ for the graph on A induced by G , and $G - A$ for the graph on $V \setminus A$ induced by G .

We say that a graph is *uniform* if all of the edges have the same weight. More specifically, we call a graph c -uniform if all edges have weight c .

A graph (V, E) is called *biconnected*, if $|V| \geq 1$, and for every vertex $v \in V$, $G - \{v\}$ is connected. A *biconnected component* of a graph is a maximal biconnected subgraph, also referred to as a *block*. It is well-known that the biconnected components of every graph partition its edges. A vertex that participates in more than one biconnected component is a *cut vertex* (usually defined as a vertex whose removal disconnects a connected component). A graph can thus be decomposed into biconnected components and cut vertices.

► **Definition 7** (Block-Cut Forest). *The block-cut forest F of a graph G has vertex set $V(F) = \mathcal{C} \cup \mathcal{B}$, where \mathcal{C} is the set of cut vertices of G and \mathcal{B} is the set of biconnected components of G , and $\{B, c\}$ is an edge in F if $B \in \mathcal{B}$, $c \in \mathcal{C}$, and $c \in V(B)$.*

It is not hard to see that the block-cut forest F of a graph G is indeed a forest, since a cycle in it would imply a cycle in G going through multiple biconnected components, thus contradicting their maximality. Moreover, each connected component of F corresponds to a connected component of G , and all leaves of F are biconnected components in G . We refer to the biconnected components of G that correspond to leaves of F as *leaf-blocks* of G .

► **Definition 8** (Uniform-Clique-Forest). *A weighted graph is a uniform-clique-forest if each of its blocks B is a uniform clique.*

► **Definition 9.** *The problem MAXCUT-WITH-VERTEX-WEIGHTS is given as follows.*

Input: *A weighted graph (V, E) with edge-weight function w , as well as two vertex-weight functions $w_0 : V \rightarrow \mathbb{N}$, $w_1 : V \rightarrow \mathbb{N}$.*

Output: *A cut C maximizing $w(C) + \sum_{v \in C} w_1(v) + \sum_{v \notin C} w_0(v)$.*

We show in Section 4 that MAXCUT-WITH-VERTEX-WEIGHTS is solvable in linear time if the input graph is a uniform-clique-forest.

3 Reducing to a Uniform-Clique-Forest

In the first part of our algorithm, we wish to either already conclude that the input graph has a cut of the desired size, or to find some set S of vertices such that $G - S$ is a uniform-clique-forest.

► **Lemma 10.** *For any graph $G = (V, E)$ on n vertices with m edges and weight function $w : E \rightarrow \mathbb{N}$ and any integer k , in time $O(n + k \cdot m)$ one can either decide that G has a cut of size at least $\frac{w(G)}{2} + \frac{w_{MSF}(G)}{4} + \frac{k}{4}$, or find a set $S \subseteq V$ such that $|S| \leq 3k$ and $G - S$ is a uniform-clique-forest.*

Note that we write $\frac{k}{4}$ instead of just k . The reason for this is that with our reduction rules we make “progress” reducing the difference to the Poljak-Turzík bound in increments of $\frac{1}{4}$.

To prove Lemma 10 we use eight reduction rules, closely inspired by the reduction rules used in [1, 5]. Each reduction rule removes some vertices from the given graph, possibly *marks* some of the removed vertices to be put into S , and possibly *reduces* the parameter k by 1. To prove Lemma 10, the reduction rules will be shown to fulfill the following properties.

Firstly, each reduction rule ensures a one-directional implication: if the reduced graph G' contains a cut of size $\frac{w(G')}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4}$ (where k' is the possibly reduced k), then the original graph G must also contain a cut of size $\frac{w(G)}{2} + \frac{w_{MSF}(G)}{4} + \frac{k}{4}$. By the Poljak-Turzík bound, if k ever reaches 0, it is clear that the original graph G must have contained a cut of the desired size.

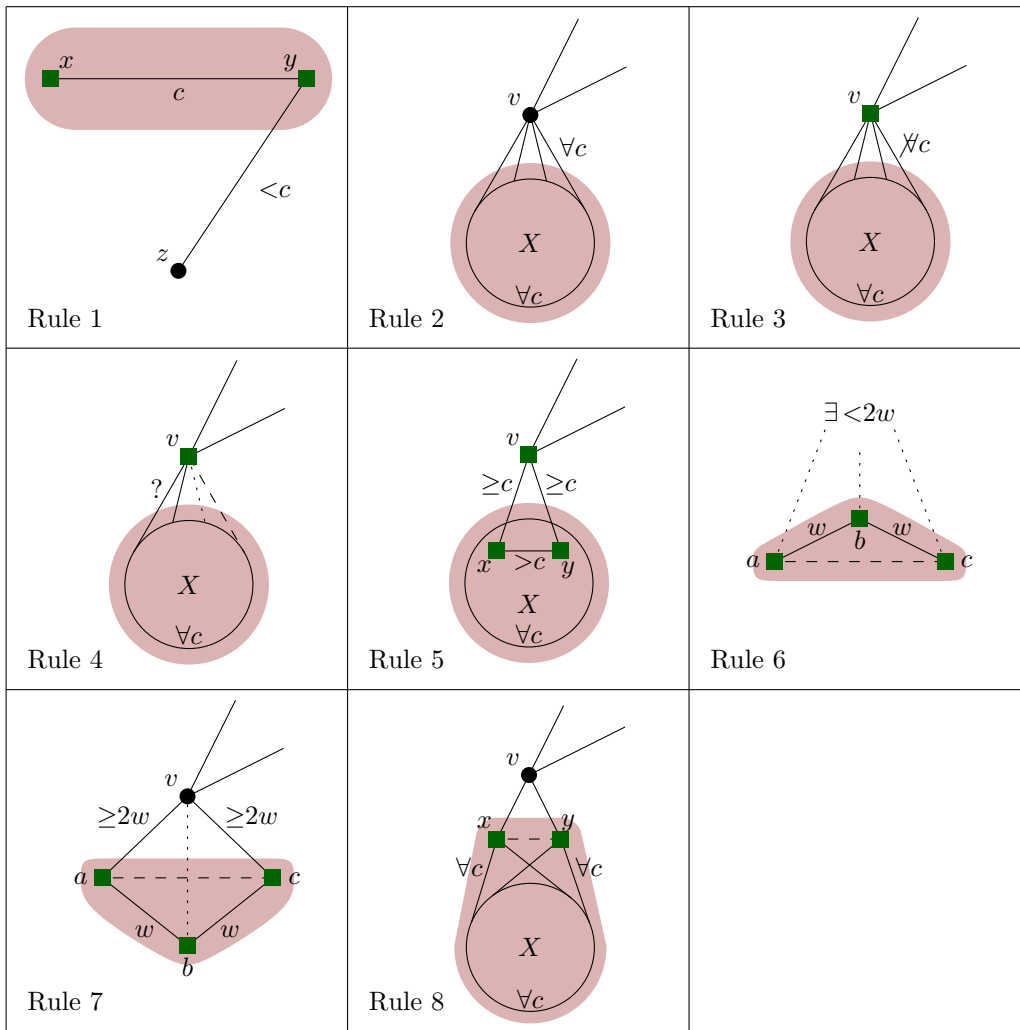
Secondly, we need that to every graph with at least one edge, at least one of the rules applies. To get our desired runtime, we also need that an applicable rule can be found and applied efficiently.

Thirdly, every rule should only mark at most three vertices to be added to S . If a rule does not reduce k , it may not mark any vertices. This ensures that at most $3k$ vertices are added to S .

Lastly, we require that after exhaustively applying the rules and reaching a graph with no more edges, the graph $G - S$ is a uniform-clique-forest.

We will now state our reduction rules, and then prove these four properties in Lemmas 11 and 12, Observation 13, and Lemma 14, respectively. For simplicity, each reduction rule is stated in such a way that it assumes the input graph to be connected. If the input graph is disconnected, instead consider G to be one of its connected components. Each rule preserves connectedness of the connected component it is applied to, which we also show in Lemma 11. Note further that if the connected component the rule is being applied to is also biconnected, then if the precondition requires some vertex to be a cut vertex, any vertex can play that role, although technically there are no cut vertices. We state this once here for simplicity, instead of saying each time that either v is a cut vertex or G is biconnected. We visualize the eight rules in Figure 1.

Rule 1:	Let $\{x, y\}, \{y, z\} \in E$ be such that $w(x, y) > w(y, z)$ and $G - \{x, y\}$ is connected.
Remove:	$\{x, y\}$
Mark:	$\{x, y\}$
Reduce k :	Yes
Rule 2:	Let $X \subseteq V, v \in V \setminus X$ be such that $X \cup \{v\}$ is a leaf-block of G with cut vertex v , and $G[X \cup \{v\}]$ is a uniform clique.
Remove:	X
Mark:	\emptyset
Reduce k :	No
Rule 3:	Let $X \subseteq V, v \in V \setminus X$ be such that $X \cup \{v\}$ is a clique and a leaf-block of G with cut vertex v ; $G[X]$ is uniform, and $G[X \cup \{v\}]$ is not uniform.
Remove:	X
Mark:	$\{v\}$
Reduce k :	Yes
Rule 4:	Let $X \subseteq V, v \in V \setminus X$ be such that $X \cup \{v\}$ is a leaf-block of G with cut vertex v ; v has at least two neighbors in X ; $G[X]$ is a uniform clique; $G[X \cup \{v\}]$ is not a clique.
Remove:	X
Mark:	$\{v\}$
Reduce k :	Yes
Rule 5:	Let $X \subseteq V, v \in V \setminus X$ be such that $X \cup \{v\}$ is a leaf-block of G with cut vertex v ; $G[X]$ is a clique; v has exactly two neighbors x, y in X ; all edges in $G[X]$ have weight c , except $\{x, y\}$, which has weight $w(x, y) > c$; $w(v, x), w(v, y) \geq c$.
Remove:	X
Mark:	$\{v, x, y\}$
Reduce k :	Yes
Rule 6:	Let $a, b, c \in V$ be such that $\{a, b\}, \{b, c\} \in E$; $\{a, c\} \notin E$; $G - \{a, b, c\}$ is connected; $w(a, b) = w(b, c)$; and $2w(a, b) > \min(\{a, b, c\}, V \setminus \{a, b, c\})$.
Remove:	$\{a, b, c\}$
Mark:	$\{a, b, c\}$
Reduce k :	Yes
Rule 7:	Let $v, a, b, c \in V$ be such that $\{a, b, c, v\}$ is a leaf-block of G with cut vertex v ; $\{a, b\}, \{b, c\}, \{a, v\}, \{c, v\} \in E$; $\{a, c\} \notin E$; $w(a, b) = w(b, c)$; $w(a, v), w(c, v) \geq 2w(a, b)$; and if $\{b, v\} \in E$ then $w(b, v) \geq 2w(a, b)$.
Remove:	$\{a, b, c\}$
Mark:	$\{a, b, c\}$
Reduce k :	Yes
Rule 8:	Let $x, y \in V$ be such that $\{x, y\} \notin E$; $G - \{x, y\}$ has exactly two connected components X and Y ; $G[X \cup \{x\}]$ and $G[X \cup \{y\}]$ are both c -uniform cliques; and x and y have exactly one neighbor v in Y .
Remove:	$X \cup \{x, y\}$
Mark:	$\{x, y\}$
Reduce k :	Yes



■ **Figure 1** The eight reduction rules. An edge is drawn normally if it must exist for the rule to apply. Some edges are drawn dashed to emphasize that they *must not* exist for the rule to apply. Some additional edges are drawn dotted to emphasize that they *may* exist but do not have to. Red shading indicates the vertices removed by the rule, while vertices marked by the rule are drawn using a green square.

We first state the formalizations of our four properties, then prove Lemma 10, and only then prove each of our properties.

► **Lemma 11.** *Let $G = (V, E)$ be a graph with weights w , and let k be any positive integer. Let G' be the result of one application of one of the rules 1–8 to G , and k' the resulting parameter. Then, if G' has a cut of size at least $\frac{w(G')}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4}$, then G must contain a cut of size at least $\frac{w(G)}{2} + \frac{w_{MSF}(G)}{4} + \frac{k}{4}$. Furthermore, if G is connected, then G' is connected.*

► **Lemma 12.** *Let $G = (V, E)$ be a weighted graph with at least one edge. Given the block-cut forest of G we can either apply Rule 2 in time $O(|E'|)$ where E' is the set of edges removed by applying Rule 2, or we can find and apply another rule in time $O(|E|)$. In the same time we can also adapt the block-cut forest.*

► **Observation 13.** *Each rule marks at most three vertices. Rule 2, the only rule that does not reduce k , does not mark any vertices.*

► **Lemma 14.** *Let S be the set of vertices marked when exhaustively (i.e., until G has no edges) applying Rules 1–8 to a graph G . Then $G - S$ is a uniform-clique-forest.*

Let us now prove Lemma 10 using these properties.

Proof of Lemma 10. We begin by computing the block-cut forest of G in $O(n + m)$ time [8]. Then, we apply rules until we either reach $k = 0$ or until we reach a graph with no edges. Whenever we apply a rule, we locally adapt the block-cut forest. In total we apply rules other than Rule 2 at most k times. By Lemma 12 this takes at most $O(k \cdot m)$ time. Since applying Rule 2 takes time $O(|E'|)$ where E' is the set of edges removed, all applications of Rule 2 together use time $O(m)$. The reduction step can thus be performed in $O(k \cdot m)$.

If we have reached $k = 0$, by the Poljak-Turzík bound and by Lemma 11 we can decide that our input graph contains a cut of the desired size. Otherwise, by Observation 13, S contains at most $3k$ vertices. By Lemma 14, $G - S$ then forms a uniform-clique-forest, and we have proven our desired statement. ◀

We will now proceed to prove Lemmas 11, 12, and 14. The main technical challenges are the proofs of Lemmas 11 and 12. These proofs are more technically involved than the corresponding proofs from [5]. For Lemma 11 this is due to the fact that the weight of a minimum spanning forest is much more difficult to track through a reduction than the number of vertices. For Lemma 12 the proof is more involved since our rules are more specific, and thus more case distinction is needed. We present the proof of Lemma 11 in Section A, since despite its technicality, it is not very insightful.

To prove Lemma 12 we use the following lemma, the proof of which follows from the proof of [5, Lemma 3] rather directly.

► **Lemma 15.** *Let $G = (V, E)$ be a connected graph with at least one edge, and let $B \subseteq V$ be a biconnected component that is a leaf in the block-cut forest of G . Now, we write B as $X \cup \{v\}$, where v is the cut vertex disconnecting $B = X \cup \{v\}$ from $V \setminus B$ (if B is an isolated vertex in the block-cut forest, i.e., it forms a connected component of G that is also biconnected, then let v be an arbitrary vertex in B). Then at least one of the following properties holds.*

- A) $G[X \cup \{v\}]$ is a clique.
- B) $G[X]$ is a clique but $G[X \cup \{v\}]$ is not a clique.
- C) v has exactly two neighbors in X , x and y . Furthermore, $\{x, y\} \notin E$, and $G[X \setminus \{x\}]$ and $G[X \setminus \{y\}]$ are cliques.
- D) $X \cup \{v\}$ contains vertices a, b, c such that $\{a, b\}, \{b, c\} \in E$, $\{a, c\} \notin E$, and $G - \{a, b, c\}$ is connected.

Furthermore, such a property (including the vertices x, y and a, b, c for cases C and D, respectively) can be found in linear time in the number of edges in $G[X]$.

Proof (sketch). One can check whether $G[X]$ is a clique for some $X \subseteq V$ in time linear in the number of edges in $G[X]$. To do this, we simply check whether each edge is present in some fixed order. It is thus easy to check for cases A), B), and C) in linear time.

In the proof of [5, Lemma 3] it is shown that if none of the cases A), B), and C) apply, then vertices a, b, c certifying case D) can be found in linear time. ◀

Proof of Lemma 12. Without loss of generality we can assume that G is connected; otherwise, we consider G to be an arbitrary connected component of our input graph that contains at least one edge. We first apply Lemma 15 on a leaf-block $X \cup \{v\}$ to find one of the four properties.

Property A) If property A) holds, we can check whether $G[X \cup \{v\}]$ is uniform in time $O(|E'|)$ where E' is the set of edges in $G[X \cup \{v\}]$. In this process we can track also whether $G[X]$ is uniform. If $G[X \cup \{v\}]$ is uniform we apply Rule 2. Else, if only $G[X]$ is uniform, we apply Rule 3. If not even $G[X]$ is uniform, we can find two edges $\{x, y\}, \{y, z\}$ in $G[X]$ such that $w(x, y) > w(y, z)$. Since $X \cup \{v\}$ is a clique, $G - \{x, y\}$ must be connected. We can therefore apply Rule 1.

Property B) We can handle property B) in a similar way. If $G[X]$ is uniform, we can apply Rule 4. Else, we apply case distinction on the number of vertices in X adjacent to v . We first consider the case if vertex v is adjacent to exactly two vertices in X . Since X is not uniform, there exist vertices $x, y \in X$ and a vertex $u \in X \cup \{v\}$ such that $w(x, y) > w(x, u)$. If the only such choice of x, y is such that x and y are exactly the two vertices in X adjacent to v , then we can apply Rule 5. Else we can see that $G - \{x, y\}$ must be connected and apply Rule 1. Let us now consider the other case, that vertex v is adjacent to at least three vertices in X . There must again exist vertices $u, x, y \in X$ so that $w(x, y) > w(x, u)$. Since v is adjacent to at least three vertices and $G[X]$ is a clique, $G - \{x, y\}$ is connected and we can apply Rule 1.

Property C) To handle Property C) we first check whether $G[X]$ is uniform. If it is not, we can apply Rule 1, since for any edge $\{a, b\}$ in $G[X]$, $G - \{a, b\}$ is connected. Knowing that $G[X]$ is uniform, and that v has exactly two neighbors, we can apply Rule 8.

Property D) Note that since $G - \{a, b, c\}$ is connected, and since by its biconnectedness $B \neq \{a, b, c\}$, if $G - B$ is non-empty, then $v \notin \{a, b, c\}$. Next, again since $G[B]$ is biconnected, we must have that $E(\{a\}, B \setminus \{a, b, c\}) \neq \emptyset$ and $E(\{c\}, B \setminus \{a, b, c\}) \neq \emptyset$. From this we get that $G - \{a, b\}$ and $G - \{b, c\}$ must be connected. Thus, we can compare $w(a, b)$ and $w(b, c)$, and apply Rule 1 if $w(a, b) \neq w(b, c)$. We can also compute the value $m = \min(\{a, b, c\}, B \setminus \{a, b, c\})$. If $2w(a, b) > m$ we can apply Rule 6.

Next, we compute the block-cut forests for the four graphs $G_{abc} := G[B] - \{a, b, c\}$ and $G_u := G[B] - \{u\}$ for all $u \in \{a, b, c\}$. This can be performed in the required time, and yields the set of cut vertices for all these graphs. We now test for every $u \in \{a, b, c\}$ and for every vertex $z \in B \setminus \{a, b, c\}$ with $\{z, u\} \in E$ whether z is a cut vertex in G_u . If for any such pair z, u we have that z is *not* a cut vertex in G_u , this means that $G - \{u, z\}$ is connected, and we can thus apply Rule 1 to that edge (recall that since we could not apply Rule 6 earlier, every edge in $E(\{a, b, c\}, B \setminus \{a, b, c\})$ has weight at least twice as large as $w(a, b) = w(b, c)$).

If every such z adjacent to some $u \in \{a, b, c\}$ is a cut vertex in G_u , we check whether any of these vertices is *not* a cut vertex in G_{abc} . If one is not, we claim that we can apply Rule 7. We prove this by distinguishing two cases, depending on u :

- $u \in \{a, c\}$: Suppose without loss of generality that $u = a$. Since z is a cut vertex of G_a , it follows that $G_a - z$ has $t \geq 2$ connected components C_1, \dots, C_t . Suppose without loss of generality that $b, c \in C_1$. If $C_1 \setminus \{b, c\} \neq \emptyset$, then $C_1 \setminus \{b, c\}$ and C_2 are two different connected components of $G_{abc} - z$, contradicting our assumption that z is not a cut vertex of G_{abc} . Thus $C_1 = \{b, c\}$, implying that b and c have no neighbours in $B \setminus \{a, b, c, z\}$. Therefore $\{c, z\} \in E$ as $E(\{c\}, B \setminus \{a, b, c\}) \neq \emptyset$, so c can also play the role of u . By symmetry, a has no neighbours in $B \setminus \{a, b, c, z\}$ and $\{a, z\} \in E$. It follows that $\{a, b, c, z\}$ is a leaf-block of G and so Rule 7 applies.
- $u = b$: Let $S := B \setminus \{a, b, c, z\}$. Recall that we established that $E(\{a\}, S \cup \{z\})$ and $E(\{c\}, S \cup \{z\})$ are both non-empty. We will show that either $\{a, z\} \in E$ or $\{c, z\} \in E$ (or both hold). Suppose that is not the case. Then $E(\{a\}, S)$ and $E(\{c\}, S)$ are both

non-empty. Since z is not a cut vertex in G_{abc} , the graph $G[S]$ must be connected. That implies $G[S \cup \{a, c\}] = G[B] - \{b, z\}$ is also connected, which contradicts our assumption that z is a cut vertex of G_b .

We have shown that at least one of $\{a, z\}$ and $\{c, z\}$ is in E , say $\{a, z\}$. Thus, without loss of generality, the case $u \in \{a, c\}$ applies, since z must be a cut vertex of G_a by our assumption that this holds for all adjacent pairs u, z with $u \in \{a, b, c\}$ and $z \in B \setminus \{a, b, c\}$.

We have thus reduced the case $u = b$ to $u \in \{a, c\}$, which we already handled.

One can now show that if this point is reached without having found an applicable rule, then Rule 7 must be applicable to the graph. Let us collect all the properties we know to be true (under the assumption that we have not found an applicable rule until now).

1. $E(\{a\}, B \setminus \{a, b, c\}) \neq \emptyset$ and $E(\{c\}, B \setminus \{a, b, c\}) \neq \emptyset$.
2. $w(a, b) = w(b, c)$
3. $2w(a, b) \leq \min(B - \{a, b, c\}, \{a, b, c\})$
4. For every pair of vertices $z \in B \setminus \{a, b, c\}$ and $u \in \{a, b, c\}$ with $\{z, u\} \in E$, z is a cut vertex of both G_u and G_{abc} .

Observe that since B is biconnected containing at most one cut vertex of G , it follows that there can be at most one cut vertex of G with a neighbor in $\{a, b, c\}$. We will now use the following claim that we will prove later.

▷ **Claim 16.** Let G be a connected graph with $X \subset G$ where X and $G - X$ are connected, and for every vertex $v \in V(G - X)$, if v has a neighbor in X , then v is a cut vertex of $G - X$. If $|N(X)| \geq 2$, then there are two distinct vertices $v_1, v_2 \in N(X)$ that are both cut vertices of G .

We apply Claim 16 on the set $X := \{a, b, c\}$. By property 4 above, and by the fact that $N(\{a, b, c\})$ contains at most one cut vertex of G , we get that $|N(\{a, b, c\})| = 1$. The vertex in $N(\{a, b, c\})$ must be the cut vertex v . By property 1 we know that $\{a, v\}, \{c, v\} \in E$. By properties 2 and 3 all the weight restrictions of Rule 7 are satisfied, which can thus be applied. ◀

Proof of Claim 16. Let H be the block-cut forest of $G - X$ and suppose $V(H) = \mathcal{C} \cup \mathcal{B}$, where \mathcal{C} are the cut vertices of $G - X$ and \mathcal{B} are the biconnected components of $G - X$. Since $|N(X)| \geq 2$, we get that $|\mathcal{C}| \geq 2$. Note that all leaves of H are in \mathcal{B} . Consider the tree T that we obtain by removing all leaves of H , and note that T has at least two vertices since $\mathcal{C} \subseteq V(T)$. Thus, T has at least two leaves, say ℓ_1, ℓ_2 , each of which must be in \mathcal{C} , since its neighbors in $H \setminus T$ are in \mathcal{B} . Let $B' \in \mathcal{B}$ be a leaf of H that is a neighbor of ℓ_i for some $i \in \{1, 2\}$. Since every vertex in $N(X)$ is in \mathcal{C} , it follows that $E(X, B' \setminus \{\ell_i\}) = \emptyset$, so ℓ_i is a cut vertex in G . ◀

For this section, it only remains to prove Lemma 14.

Proof of Lemma 14. Let G_1, G_2, \dots, G_q be the sequence of graphs obtained while exhaustively applying rules 1–8 to G_1 (G_2 is the graph obtained after applying one rule to G_1 , G_3 is the graph obtained after applying one rule to G_2 , and so on). We prove that for any graph G_i in the sequence, $G_i - S$ is a uniform-clique-forest. We run this proof by induction over the sequence of graphs in reverse order (in the order $G_q, G_{q-1}, \dots, G_2, G_1$).

Base Case: By Lemma 12, we know that G_q is a graph without edges, therefore $G_q = G_q - S$ is trivially a uniform-clique-forest.

Induction Hypothesis: Assume $G_i - S$ is a uniform-clique-forest.

Step Case: We prove that $G_{i-1} - S$ is a uniform-clique-forest. We know that one rule among rules 1–8 was applied to G_{i-1} to obtain G_i . We do a case distinction over which rule was applied:

- Rule 1, 6, or 7 was applied to G_{i-1} . Every vertex these rules remove is also marked, therefore $G_{i-1} - S = G_i - S$.
- Rule 2 was applied to G_{i-1} . We can create $G_{i-1} - S$ from $G_i - S$ by connecting a clique X to a vertex $v \in V(G_i)$ such that $X \cup \{v\}$ is a uniform clique. If v is in S , this is instead adding a disjoint uniform clique. Observe that this just adds a uniform leaf-clique in either case.
- Rule 3, 4, 5, or 8 was applied to G_{i-1} . We can create $G_{i-1} - S$ from $G_i - S$ by adding a disjoint uniform clique.

We conclude that in all cases $G_{i-1} - S$ consists of one or zero uniform cliques added to $G_i - S$ as a leaf, and thus by the induction hypothesis $G_{i-1} - S$ is a uniform-clique-forest. ◀

4 Solving MaxCut-With-Vertex-Weights on Uniform-Clique-Forests

► **Lemma 17.** *MAXCUT-WITH-VERTEX-WEIGHTS on a uniform-clique-forest G with n vertices and m edges can be solved in $O(n + m)$ time.*

Proof. This proof loosely follows the proof of [5, Lemma 4]. We first compute the cut-block forest of G . We know that every graph contains at least one leaf-block. Let $X \cup \{v\}$ be a leaf-block of G where $v \in V(G)$ is the cut vertex of X (if a connected component of G consists of a single biconnected component B , then $X = B - \{v\}$ where v is an arbitrary vertex in B). Let $n' = |X|$ and m' be the number of edges in $G[X \cup \{v\}]$. Since G is a uniform-clique-forest, we know that $G[X \cup \{v\}]$ is c -uniform for some c . We now consider the maximum weighted cut in $G[X \cup \{v\}]$ for both possible cases $v \notin C$ and $v \in C$.

We first consider $v \notin C$. Let $\delta(x) = w_1(x) - w_0(x)$ for every vertex $x \in X$. We can sort the vertices in X in the order $x_1, x_2, \dots, x_{n'}$ with decreasing δ -value, i.e., $\delta(x_1) \geq \dots \geq \delta(x_{n'})$. For any $p \in \{0, \dots, n'\}$, we let A_p be the set $\{x_1, \dots, x_p\}$. Clearly A_p is the best cut among all cuts C' with $|C' \cap X| = p$. Now we can find the maximum weighted cut in $X \cup \{v\}$ by comparing the $n' + 1$ cuts $A_0, \dots, A_{n'}$. Letting λ be the value of this cut, we update $w_0(v) = \lambda$.

We can perform the same process for $v \in C$. We instead consider $A_p = \{v, x_1, \dots, x_p\}$, and update $w_1(v)$ to the optimum value found. After having updated both weights for v , we can now delete all vertices in X .

We can apply this method to G exhaustively until we are left with a graph with no edges. The desired value of the maximum weighted cut on the entire graph G is the sum of the greater values of $w_0(v)$ or $w_1(v)$ for all remaining vertices v .

We now calculate the runtime of this method applied to one leaf-block X . Sorting the vertices takes $O(n' \log(n'))$ time. Since X is a clique, we have $n' \log(n') \leq \frac{n'(n'+1)}{2} = m'$ for all $n' \geq 4$. We can calculate the value of the assignment A_0 in $O(m')$ time. Observe that the difference between cuts A_i and A_{i+1} for any $i \in \{0, \dots, n-1\}$ is in only one vertex. By only considering these local modifications we can calculate the values of the cuts $A_0, \dots, A_{n'}$ in $O(m')$ time. Since in every iteration we perform this process on a different block, in total we can bound our runtime with $O(n + m)$, since for blocks with $n' < 4$ the runtime of $O(n' \log(n')) = O(1)$ can be charged to some vertex in the block, while for blocks with $n' \geq 4$ the runtime of $O(n' \log(n'))$ can be expressed as $O(m')$. ◀

5 Conclusion

With Lemmas 10 and 17, our main result now follows easily:

Proof of Theorem 4. Given any instance $\text{MAXCUT}(G, \frac{w(G)}{2} + \frac{w_{MSF}(G)}{4} + \frac{k'}{4})$ with $k' := 4k$, by Lemma 10 we can in time $O(n + k \cdot m)$ either decide that the instance is a “yes”-instance, or find a set $S \subseteq V$ with $|S| \leq 3k' = 12k$ such that $G - S$ is a uniform-clique-forest. For each subset $S' \subseteq S$ we can then in time $O(n + m)$ build a $\text{MAXCUT-WITH-VERTEX-WEIGHTS}$ instance on the graph $G - S$, such that the vertex weights $w_0(v)$ and $w_1(v)$ of a vertex $v \in G - S$ denote the sum of the weights of edges to vertices in S' and $S \setminus S'$ respectively. By Lemma 17, each of these instances can be solved in $O(n + m)$ time. The maximum cut found in any instance given by a set S' corresponds to the maximum cut C of G obtainable under the condition that $C \cap S = S'$. Taking into account the edges between S and S' and taking the maximum over all instances thus computes the maximum cut size of G .

To compute the overall runtime, note that since $|S| \leq 12k$, we solve at most 2^{12k} $\text{MAXCUT-WITH-VERTEX-WEIGHTS}$ instances. Thus, the overall runtime is $O(n + k \cdot m + 2^{O(k)} \cdot (n + m)) = O(2^{O(k)} \cdot (n + m))$. ◀

If we want to find a cut instead of deciding the existence of a cut, we can use very similar techniques.

Proof of Theorem 5. The proof of Lemma 11 is constructive: given a cut C' on the reduced graph G' of the assumed size, a cut C on the original graph G of the required size can be found in linear time in the number of removed edges and vertices. Thus, instead of applying reduction rules only until $k \leq 0$ or until the graph has no edges, we *always* apply rules until the graph contains no edges. This requires at most $O(n \cdot m)$ time. Note that when we have removed all edges from the graph, the required size of a cut (k larger than the Poljak-Turzík bound) is simply $\frac{0}{2} + \frac{0}{4} + k = k$. Thus, if $k \leq 0$ is reached, the required cut size is non-positive, thus we can start with any arbitrary cut C' of the remaining independent set. We can then apply the cut extensions from the proof of Lemma 11 for all applied rules in reverse. This yields a cut of G of the desired size. If otherwise we have $k > 0$ when we reached a graph with no edges, we know that $|S| \leq 12k$, and we can again solve $2^{|S|}$ instances of $\text{MAXCUT-WITH-VERTEX-WEIGHTS}$ on $G - S$. ◀

Open Problems

Our result leaves a few interesting open problems.

Other λ -extendible properties. In [13], Poljak and Turzík actually not only show the lower bound for MAXCUT (Theorem 3) but in fact they prove a very similar bound for the existence of large subgraphs fulfilling any so-called λ -*extendible* property.² Mnich, Philip, Saurabh, and Suchý [12] generalize the approach of [1] for MAXCUT to work for a large subset of these λ -extendible properties. Note that while the title of [12] includes “above the Poljak-Turzík bound”, the authors restrict their attention to unweighted simple graphs, and thus their result applied to MAXCUT only implies the result of [1], but *not* our result. We find it a very interesting direction to see if our result can be extended to also cover some more λ -extendible properties in multigraphs or positive integer-weighted graphs.

² For MAXCUT this property would be bipartiteness.

Kernelization. Many previous works on MAXCUT parameterized above guaranteed lower bounds have also provided kernelization results [1, 5, 10]. In particular, together with their linear-time algorithm parameterized by the distance k to the Poljak-Turzík bound, Etscheid and Mních [5] also provide a linear-sized (in k) kernel. We are not aware of any kernelization results for MAXCUT on multigraphs or positive integer-weighted graphs. It would thus be very interesting to explore whether these results can also be extended to our setting.

FPT above better lower bounds. Recently, Gutin and Yeo [6] proved new lower bounds for $\mu(G)$ for positive real-weighted graphs. In particular, they prove $\mu(G) \geq \frac{w(G)}{2} + \frac{w(M)}{2}$ where M is a maximum matching of G , and $\mu(G) \geq \frac{w(G)}{2} + \frac{w(D)}{4}$ for any DFS-tree D (which implies the Poljak-Turzík bound). Both of these bounds are consequences of a more general bound involving disjoint bipartite induced subgraphs, but the value of this bound is NP-hard to compute [6]. The weight of the largest DFS-tree is also NP-hard to compute [6]. These two bounds are thus not very suitable for an FPT algorithm, but the bound involving the maximum matching may be, since the maximum matching in a weighted graph can be computed in polynomial time using Edmonds' blossom algorithm.

General weights. After going from simple graphs to multigraphs and thus positive integer-weighted graphs, it would be interesting to further generalize to positive real-weighted graphs. Here, it is not directly clear what the parameter k exactly should be. Generalizing our algorithm may require completely new approaches since we cannot discretize the decrease of k .

References

- 1 Robert Crowston, Mark Jones, and Matthias Mních. Max-cut parameterized above the Edwards-Erdős bound. *Algorithmica*, 72(3):734–757, 2015. doi:10.1007/s00453-014-9870-z.
- 2 Christopher S. Edwards. Some extremal properties of bipartite subgraphs. *Canadian Journal of Mathematics*, 25(3):475–485, 1973. doi:10.4153/CJM-1973-048-x.
- 3 Christopher S. Edwards. An improved lower bound for the number of edges in a largest bipartite subgraph. In *Proc. 2nd Czechoslovak Symposium on Graph Theory, Prague*, pages 167–181, 1975.
- 4 Paul Erdős. On some extremal problems in graph theory. *Israel Journal of Mathematics*, 3(2):113–116, 1965. doi:10.1007/BF02760037.
- 5 Michael Etscheid and Matthias Mních. Linear kernels and linear-time algorithms for finding large cuts. *Algorithmica*, 80(9):2574–2615, 2018. doi:10.1007/s00453-017-0388-z.
- 6 Gregory Gutin and Anders Yeo. Lower bounds for maximum weighted cut. *SIAM Journal on Discrete Mathematics*, 37(2):1142–1161, 2023. doi:10.1137/21M1411913.
- 7 David J. Haglin and Shankar M. Venkatesan. Approximation and intractability results for the maximum cut problem and its variants. *IEEE Transactions on Computers*, 40(1):110–113, 1991. doi:10.1109/12.67327.
- 8 John Hopcroft and Robert Tarjan. Algorithm 447: efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378, 1973. doi:10.1145/362248.362272.
- 9 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, editors, *Complexity of Computer Computations*, pages 85–103, Boston, MA, 1972. Springer US. doi:10.1007/978-1-4684-2001-2_9.
- 10 Jayakrishnan Madathil, Saket Saurabh, and Meirav Zehavi. Fixed-parameter tractable algorithm and polynomial kernel for max-cut above spanning tree. *Theory of Computing Systems*, 64(1):62–100, 2020. doi:10.1007/s00224-018-09909-5.
- 11 Meena Mahajan and Venkatesh Raman. Parameterizing above guaranteed values: Maxsat and maxcut. *Journal of Algorithms*, 31(2):335–354, 1999. doi:10.1006/jagm.1998.0996.

- 12 Matthias Mnich, Geevarghese Philip, Saket Saurabh, and Ondřej Suchý. Beyond max-cut: λ -extendible properties parameterized above the Poljak–Turzík bound. *Journal of Computer and System Sciences*, 80(7):1384–1403, 2014. doi:10.1016/j.jcss.2014.04.011.
- 13 Svatopluk Poljak and Daniel Turzík. A polynomial time heuristic for certain subgraph optimization problems with guaranteed worst case bound. *Discrete Mathematics*, 58(1):99–104, 1986. doi:10.1016/0012-365X(86)90192-5.
- 14 Svatopluk Poljak and Zsolt Tuza. Maximum cuts and large bipartite subgraphs. *DIMACS Series*, 20:181–244, 1995. doi:10.1090/dimacs/020/04.

A Proof of Lemma 11

We will often use the following claim that slightly strengthens the Poljak-Turzík bound in certain cases:

▷ **Claim 18.** Let $G = (V, E)$ be a weighted graph with weights $w : E \rightarrow \mathbb{N}$ such that there exist edges $\{u, v\}, \{v, x\}$ with $w(u, v) > w(v, x)$ and $G - \{u, v\}$ is connected. Then G has a cut of size at least $\frac{w(G)}{2} + \frac{w_{MSF}(G)}{4} + \frac{1}{4}$.

Proof. Let $G' = G - \{u, v\}$. By the Poljak-Turzík bound we know we have a cut C' of G' of size at least $\frac{w(G')}{2} + \frac{w_{MSF}(G')}{4}$. We can extend this to a cut C in G by adding exactly one of u and v . We choose the one such that at least half of the weight in $E(\{u, v\}, V')$ goes over the cut. We have that $MSF(G') \cup \{u, v\} \cup \{v, x\}$ is a spanning forest of G , therefore $w_{MSF}(G') + w(u, v) + w(v, x) \geq w_{MSF}(G)$. The cut C has weight at least

$$\begin{aligned}
 w(C) &\geq \frac{w(G')}{2} + \frac{w_{MSF}(G')}{4} + \frac{w(\{u, v\}, V')}{2} + w(u, v) \\
 &= \frac{w(G)}{2} + \frac{w_{MSF}(G')}{4} + \frac{w(u, v)}{2} \\
 &\geq \frac{w(G)}{2} + \frac{w_{MSF}(G')}{4} + \frac{w(u, v)}{4} + \frac{w(v, x) + 1}{4} \\
 &\geq \frac{w(G)}{2} + \frac{w_{MSF}(G)}{4} + \frac{1}{4}. \quad \triangleleft
 \end{aligned}$$

Let us now prove Lemma 11.

Proof of Lemma 11. We first see that each rule preserves connectedness simply by their preconditions. Each rule either explicitly requires that the resulting graph is connected (Rules 1, 6, and 8), or removes a whole leaf-block of G , except for the cut vertex (Rules 2–5 and 7).

We now prove the required cut size implication for each rule independently. We need to prove that if there exists a cut C' in G' that produces a cut of size $\frac{w(G')}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4}$, then this can be extended to a cut C of G of size $\frac{w(G)}{2} + \frac{w_{MSF}(G)}{4} + \frac{k}{4}$. We thus assume that such a cut C' exists, and then extend it in such a way that $C \cap V' = C'$. We perform a case distinction on the rule that we applied to G to obtain G' . Recall that for all rules except Rule 2, $k' = k - 1$.

Rule 1: We extend C' by putting x and y on different sides of the cut. Among the two possibilities, we choose the one such that at least half the weight in $E(\{x, y\}, V')$ goes over the cut. We get a cut of size at least

$$\begin{aligned}
w(C') &\geq \frac{w(G')}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} + w(x, y) + \frac{w(E(\{x, y\}, V'))}{2} \\
&= \frac{w(G)}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} + \frac{w(x, y)}{2} \\
&\geq \frac{w(G)}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} + \frac{w(x, y)}{4} + \frac{w(y, z) + 1}{4}
\end{aligned}$$

We now see that $w_{MSF}(G) \leq w_{MSF}(G') + w(x, y) + w(y, z)$, and we thus get

$$w(C') \geq \frac{w(G)}{2} + \frac{w_{MSF}(G)}{4} + \frac{k}{4}.$$

Rule 2: We can assume without loss of generality that $v \in C'$. Let $n' := |X \cup \{v\}|$. Observe that the sum of the total weight in $G[X \cup \{v\}]$ is $c \binom{n'(n'-1)}{2}$ for the integer c such that all edges in $G[X \cup \{v\}]$ have weight c . If n' is odd, $|X|$ is even, and we can add exactly half the vertices to C . This way we have a cut C'' in $G[X \cup \{v\}]$ of size at least

$$\begin{aligned}
w(C'') &\geq c \binom{\frac{n'+1}{2}}{2} \binom{\frac{n'-1}{2}}{2} \\
&= c \binom{\frac{n'(n'-1)}{4}}{2} + c \binom{\frac{n'-1}{4}}{2} \\
&= \frac{w(G[X \cup \{v\}])}{2} + \frac{w_{MSF}(G[X \cup \{v\}])}{4}.
\end{aligned}$$

If n' is even, we add $\frac{n'}{2}$ of the vertices of X to C , and leave $\frac{n'}{2} - 1$ vertices out of C . In this case, we have a cut C'' in $G[X \cup \{v\}]$ of size at least

$$\begin{aligned}
w(C'') &\geq c \binom{\frac{n'}{2}}{2} \binom{\frac{n'}{2}}{2} \\
&= c \binom{\frac{n'(n'-1)}{4}}{2} + c \binom{\frac{n'}{4}}{2} \\
&\geq \frac{w(G[X \cup \{v\}])}{2} + \frac{w_{MSF}(G[X \cup \{v\}])}{4}.
\end{aligned}$$

In either case, we can see that we can combine C' and C'' to a cut C in G of size at least

$$\begin{aligned}
w(C) &\geq \frac{w(G[X \cup \{v\}])}{2} + \frac{w_{MSF}(G[X \cup \{v\}])}{4} + \frac{w(G')}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} \\
&= \frac{w(G)}{2} + \frac{w_{MSF}(G)}{4} + \frac{k}{4},
\end{aligned}$$

where in the last equality we used that $k' = k$ for this rule.

Rule 3: Since $G[X \cup \{v\}]$ is not uniform, we can apply Claim 18 to $G[X \cup \{v\}]$ to obtain a cut C'' in $G[X \cup \{v\}]$ of size at least $\frac{w(G[X \cup \{v\}])}{2} + \frac{w_{MSF}(G[X \cup \{v\}])}{4} + \frac{1}{4}$. We now assume without loss of generality that $v \in C'' \Leftrightarrow v \in C'$, i.e., both C' and C'' put v on the same side of the cut. In this case we can combine C' and C'' to a cut C of size at least

$$\begin{aligned}
w(C) &\geq \frac{w(G[X \cup \{v\}])}{2} + \frac{w_{MSF}(G[X \cup \{v\}])}{4} + \frac{1}{4} + \frac{w(G')}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} \\
&= \frac{w(G)}{2} + \frac{w_{MSF}(G)}{4} + \frac{k}{4}.
\end{aligned}$$

Rule 4: We know that v must be adjacent to more than 1 and less than $|X|$ vertices of X . We first do a case distinction on whether $G[X \cup \{v\}]$ is uniform or not.

If $G[X \cup \{v\}]$ is not uniform, we use the same argument as for the previous rule. Let $y \in X$ be a vertex not adjacent to v . Observe that for any $x \in X$ such that $\{v, x\} \in E$, $G[X \cup \{v\} - \{v, x\}]$ and $G[X \cup \{v\} - \{x, y\}]$ are both connected. Since $G[X \cup \{v\}]$ is not uniform but $G[X]$ is, we can find such an x such that either $w(x, y) > w(x, v)$ or $w(x, y) < w(x, v)$. Therefore, we can use Claim 18 on $G[X \cup \{v\}]$. This gives us a cut C'' in $G[X \cup \{v\}]$ of size at least $\frac{w(G[X \cup \{v\}])}{2} + \frac{w_{MSF}(G[X \cup \{v\}])}{4} + \frac{1}{4}$. Combining this cut with C' , we get a cut C in G of size at least

$$\begin{aligned} w(C) &\geq \frac{w(G')}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} + \frac{w(G[X \cup \{v\}])}{2} + \frac{w_{MSF}(G[X \cup \{v\}])}{4} + \frac{1}{4} \\ &\geq \frac{w(G)}{2} + \frac{w_{MSF}(G)}{4} + \frac{k}{4}. \end{aligned}$$

Otherwise $G[X \cup \{v\}]$ is c -uniform. Let $m' = w(G[X \cup \{v\}])$ and $n' = |X|$. We can order the vertices in X as $x_1, x_2, \dots, x_{n'}$ such that v is adjacent to exactly x_1, \dots, x_r , but not $x_{r+1}, \dots, x_{n'}$. Assume without loss of generality that $v \in C'$. We add v and all x_i for $i > \lceil \frac{n'}{2} \rceil$ to a cut C'' of $G[X \cup \{v\}]$. This cut has size $s := c(\lceil \frac{n'}{2} \rceil \cdot \lfloor \frac{n'}{2} \rfloor + \min\{r, \lceil \frac{n'}{2} \rceil\})$. Note that $m' = c(\frac{n'(n'-1)}{2} + r)$, thus we can rephrase $s = \frac{m'}{2} + c(\frac{n'}{4} - \frac{n'^2}{4} + (\lceil \frac{n'}{2} \rceil)(\lfloor \frac{n'}{2} \rfloor) + \min\{\frac{r}{2}, \lceil \frac{n'}{2} \rceil - \frac{r}{2}\})$. If n' is even, $(\lceil \frac{n'}{2} \rceil)(\lfloor \frac{n'}{2} \rfloor) = \frac{n'^2}{4}$, and then $s \geq \frac{m'}{2} + c\frac{n'}{4} + \frac{c}{2} \geq \frac{m'}{2} + c\frac{n'}{4} + \frac{1}{4}$. If n' is odd, $(\lceil \frac{n'}{2} \rceil)(\lfloor \frac{n'}{2} \rfloor) = \frac{(n'+1)(n'-1)}{4} = \frac{n'^2}{4} - \frac{1}{4}$, and then $s \geq \frac{m'}{2} + c\frac{n'}{4} + \frac{c}{2} - \frac{c}{4} \geq \frac{m'}{2} + c\frac{n'}{4} + \frac{1}{4}$, as well.

In either case we can combine C'' on $G[X \cup \{v\}]$ and C' on G' to get a cut C of G of size at least

$$\begin{aligned} w(C) &\geq \frac{w(G')}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} + \frac{m'}{2} + c\frac{n'}{4} + \frac{1}{4} \\ &\geq \frac{w(G)}{2} + \frac{w_{MSF}(G)}{4} + \frac{k}{4}, \end{aligned}$$

where we used that an MSF of G' can be turned into a spanning forest of G by adding n' edges of weight c .

Rule 5: Let $X' = X - \{x, y\}$. If $w(x, v) > c$ or $w(y, v) > c$, since $G[X \cup \{v\} - \{v, x\}]$ and $G[X \cup \{v\} - \{v, y\}]$ are connected, we know by Claim 18 that $G[X \cup \{v\}]$ has a cut C'' of size at least $\frac{w(G[X \cup \{v\}])}{2} + \frac{w_{MSF}(G[X \cup \{v\}])}{4} + \frac{1}{4}$. Since G' and $G[X \cup \{v\}]$ overlap in only one vertex v we can w.l.o.g. assume that $v \in C'' \Leftrightarrow v \in C'$, and we can combine C'' and C' to a cut C of G of size at least

$$\begin{aligned} w(C) &\geq \frac{w(G')}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} + \frac{w(G[X \cup \{v\}])}{2} + \frac{w_{MSF}(G[X \cup \{v\}])}{4} + \frac{1}{4} \\ &\geq \frac{w(G)}{2} + \frac{w_{MSF}(G)}{4} + \frac{k}{4}. \end{aligned}$$

Thus, from now on we may assume $w(x, v) = w(y, v) = c$, i.e., the only edge in $G[X \cup \{v\}]$ that does not have weight c is the edge $\{x, y\}$ of weight $> c$. For the remaining cases, we perform a case distinction over the size of X' . Without loss of generality we assume that $v \notin C'$.

- Case 1: $|X'| = 1$. Let u be the only vertex in X' . Observe $w_{MSF}(G') + w(x, v) + w(y, v) + w(u, x) \geq w_{MSF}(G)$.

- Assume $w(x, y) > 2c$. We create C by adding x to C' . Then C has size at least

$$\begin{aligned}
w(C) &\geq \frac{w(G')}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} + w(x, y) + w(x, v) + w(x, u) \\
&\geq \frac{w(G')}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} + \frac{w(x, y) + w(x, v) + w(x, u) - w(y, v) - w(y, u)}{2} \\
&= \frac{w(G')}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} + \frac{w(x, y)}{2} \\
&> \frac{w(G')}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} + \frac{2c}{2} \\
&\geq \frac{w(G')}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} + \frac{3c}{4} + \frac{1}{4} \\
&\geq \frac{w(G')}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4}.
\end{aligned}$$

- Assume $w(x, y) \leq 2c$. We create C by adding x and y to C' . Then C has size at least

$$\begin{aligned}
w(C) &\geq \frac{w(G')}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} + w(y, v) + w(y, u) + w(x, v) + w(x, u) \\
&\geq \frac{w(G')}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} + 3c + \frac{w(x, y)}{2} \\
&\geq \frac{w(G')}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} + c \\
&\geq \frac{w(G')}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} + \frac{c}{4} \\
&\geq \frac{w(G')}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4}.
\end{aligned}$$

- Case 2: $|X'| =: n' > 1$. Observe $w(G) = w(G') + c\binom{n'(n'-1)}{2} + 2n' + 2 + w(x, y)$ and $w_{MSF}(G') + c(n' + 2) \geq w_{MSF}(G)$.

- Assume $w(x, y) \geq 2c$. We start with a cut on $G[X']$ of size at least $\frac{w(G[X'])}{2} + \frac{w_{MSF}(G[X'])}{4} = \frac{w(G[X'])}{2} + c\binom{n'-1}{4}$ as guaranteed by the Poljak-Turzík bound. Then we extend this to a cut on $G[X' \cup \{x, y\}]$ by adding exactly one of x and y , choosing of the two possibilities the one that cuts at least half the weight in $E(X', \{x, y\})$. We combine this cut with the cut C' .

The resulting cut C has size at least

$$\begin{aligned}
w(C) &\geq \frac{w(G')}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} + \frac{w(G[X'])}{2} + c\binom{n'-1}{4} + \frac{|E(X', \{x, y\})|}{2} + w(x, y) + c \\
&\geq \frac{w(G')}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} + c\binom{n'-1}{4} + \frac{w(x, y)}{2} \\
&\geq \frac{w(G')}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} + c\binom{n'+3}{4} \\
&\geq \frac{w(G')}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} + \frac{c}{4} \\
&\geq \frac{w(G')}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4}.
\end{aligned}$$

- Assume $w(x, y) < 2c$. We add both x and y to the cut C' .

If n' is odd we add $\frac{n'-1}{2}$ vertices of X' to C' . The resulting cut C has size at least

$$\begin{aligned}
 w(C) &\geq \frac{w(G')}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} + c \left(\binom{n'+3}{2} \binom{n'+1}{2} + 2 \right) \\
 &\geq \frac{w(G')}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} + c \left(\frac{n'^2 + 4n' + 3}{4} + 1 \right) + \frac{w(x,y) + 1}{2} \\
 &= \frac{w(G)}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} + c \left(\frac{n'-1}{4} + 1 \right) + \frac{1}{2} \\
 &= \frac{w(G)}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} + c \left(\frac{n'+3}{4} \right) + \frac{1}{2} \\
 &\geq \frac{w(G)}{2} + \frac{w_{MSF}(G)}{4} + \frac{k'}{4} + \frac{1}{2} \\
 &\geq \frac{w(G)}{2} + \frac{w_{MSF}(G)}{4} + \frac{k}{4}.
 \end{aligned}$$

If n' is even we add $\frac{n'}{2} - 1$ vertices of X' to C' . The resulting cut C has size at least

$$w(C) \geq \frac{w(G')}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} + c \left(\binom{n'+2}{2} \binom{n'+2}{2} + 2 \right),$$

which is strictly larger than in the n' odd case.

Rule 6: Let $\min = \min(V \setminus \{a, b, c\}, \{a, b, c\})$, and let e_{\min} be an edge of weight \min in $E(V \setminus \{a, b, c\}, \{a, b, c\})$. Observe that $MSF(G')$ together with e_{\min} , $\{a, b\}$, and $\{b, c\}$ forms a spanning forest of G . Therefore $w_{MSF}(G') + \min + w(a, b) + w(b, c) = w_{MSF}(G') + \min + 2w(a, b) \geq w_{MSF}(G)$.

We consider two subsets of $\{a, b, c\}$: $A_1 = \{a, c\}$, and $A_2 = \{b\}$. Considering these as cuts of G , both cuts cut the edges $\{a, b\}$ and $\{b, c\}$, and at least one of these cuts gets at least half of the total weight in $E(V(G'), \{a, b, c\})$. Enhancing C' by that set, we therefore get a cut C of G of size at least

$$\begin{aligned}
 w(C) &\geq \frac{w(G')}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} + \frac{w(V(G'), \{a, b, c\})}{2} + w(a, b) + w(b, c) \\
 &\geq \frac{w(G)}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} + \frac{w(a, b)}{2} + \frac{w(b, c)}{2} \\
 &\geq \frac{w(G)}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} + \frac{w(a, b)}{2} + \frac{\min + 1}{4} \\
 &\geq \frac{w(G)}{2} + \frac{w_{MSF}(G)}{4} + \frac{k'}{4} + \frac{1}{4} \\
 &\geq \frac{w(G)}{2} + \frac{w_{MSF}(G)}{4} + \frac{k}{4}.
 \end{aligned}$$

Rule 7: If b is adjacent to v we can augment C' by adding a, b, c to C if and only if $v \notin C'$. Observe $MSF(G') \cup \{a, v\} \cup \{b, v\} \cup \{c, v\}$ is a spanning forest of G . Also by the conditions of Rule 7 we have $\frac{w(a,v)}{4} + \frac{w(b,v)}{4} \geq \frac{w(a,b)}{2} + \frac{w(a,b)}{2} = \frac{w(a,b)}{2} + \frac{w(b,c)}{2}$. We can thus analyze the cut C to have size at least

$$\begin{aligned}
w(C) &\geq \frac{w(G')}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} + w(a, v) + w(b, v) + w(c, v) \\
&\geq \frac{w(G')}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} + \frac{3w(a, v)}{4} + \frac{3w(b, v)}{4} + \frac{w(a, b)}{2} + \frac{w(b, c)}{2} + w(c, v) \\
&\geq \frac{w(G)}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} + \frac{w(a, v)}{4} + \frac{w(b, v)}{4} + \frac{w(c, v)}{2} \\
&\geq \frac{w(G)}{2} + \frac{w_{MSF}(G)}{4} + \frac{k'}{4} + \frac{w(c, v)}{4} \\
&\geq \frac{w(G)}{2} + \frac{w_{MSF}(G)}{4} + \frac{k}{4}.
\end{aligned}$$

If b is not adjacent to v , add a, c to C if and only if $v \notin C'$, and we add b to C if and only if $v \in C'$. Thus the edges $\{a, b\}, \{b, c\}, \{a, v\}, \{c, v\}$ are all cut. Note that $MSF(G') \cup \{c, v\} \cup \{a, b\} \cup \{b, c\}$ is a spanning forest of G . The cut C has size at least

$$\begin{aligned}
w(C) &\geq \frac{w(G')}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} + w(a, v) + w(a, b) + w(b, c) + w(c, v) \\
&\geq \frac{w(G)}{2} + \frac{w_{MSF}(G)}{4} + \frac{k'}{4} + \frac{w(a, v)}{2} + \frac{w(c, v)}{4} + \frac{w(a, b)}{4} + \frac{w(b, c)}{4} \\
&\geq \frac{w(G)}{2} + \frac{w_{MSF}(G)}{4} + \frac{k}{4}.
\end{aligned}$$

Rule 8: Let v be the only neighbor of $\{x, y\}$ in Y and let $\bar{n} = |X|$. We first extend C' to C'' by adding x, y to C'' if and only if $v \notin C'$. We then extend C'' to C as follows.

Without loss of generality, assume $x, y \notin C''$. We perform a case distinction on the parity of \bar{n} . Note that $w(G[X \cup \{x, y\}]) = c\left(\frac{\bar{n}(\bar{n}-1)}{2} + 2\bar{n}\right)$.

- If \bar{n} is odd, we add $\frac{\bar{n}+1}{2}$ of the vertices in X to C . In $G[X \cup \{x, y\}]$ this cuts in total a weight of

$$c\left(\left(\frac{\bar{n}+1}{2}\right)\left(\frac{\bar{n}-1}{2}\right) + 2\frac{\bar{n}+1}{2}\right) = c\left(\frac{\bar{n}(\bar{n}-1)}{4} + \frac{\bar{n}-1}{4} + \bar{n} + 1\right) = \frac{w(G[X \cup \{x, y\}])}{2} + c\left(\frac{\bar{n}}{4} + \frac{3}{4}\right).$$

- If \bar{n} is even, we add $\frac{\bar{n}}{2} + 1$ vertices in X to C . In $G[X \cup \{x, y\}]$ this cuts in total a weight of

$$c\left(\left(\frac{\bar{n}}{2} + 1\right)\left(\frac{\bar{n}}{2} - 1\right) + 2\left(\frac{\bar{n}}{2} + 1\right)\right) = c\left(\frac{\bar{n}^2}{4} + \bar{n} + 1\right) = c\left(\frac{\bar{n}^2}{4} + \frac{3}{4}\bar{n} + \frac{\bar{n}}{4} + 1\right) = \frac{w(G[X \cup \{x, y\}])}{2} + c\left(\frac{\bar{n}}{4} + 1\right).$$

In either case we thus have that C cuts at least half of the weight in $G[X \cup \{x, y\}]$ plus $c\left(\frac{\bar{n}}{4} + \frac{3}{4}\right)$.

Observe that $w_{MSF}(G) \leq w_{MSF}(G') + c\bar{n} + w(x, v) + w(y, v)$. In total we can thus bound the size of the cut C as

$$\begin{aligned}
w(C) &\geq \frac{w(G')}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} + \frac{w(G[X \cup \{x, y\}])}{2} + c\left(\frac{\bar{n}}{4} + \frac{3}{4}\right) + w(x, v) + w(y, v) \\
&= \frac{w(G)}{2} + \frac{w_{MSF}(G')}{4} + \frac{k'}{4} + c\left(\frac{\bar{n}}{4} + \frac{3}{4}\right) + \frac{w(x, v)}{2} + \frac{w(y, v)}{2} \\
&\geq \frac{w(G)}{2} + \frac{w_{MSF}(G)}{4} + \frac{k}{4}.
\end{aligned}$$

We conclude that for every rule, from a cut C' of G' of the guaranteed size we can build a cut C of G of the required size, and thus the lemma follows. ◀

Twin-Width Meets Feedback Edges and Vertex Integrity

Jakub Balabán  

Faculty of Informatics, Masaryk University, Brno, Czech Republic

Robert Ganian  

Algorithms and Complexity Group, TU Wien, Vienna, Austria

Mathis Rocton  

Algorithms and Complexity Group, TU Wien, Vienna, Austria

Abstract

The approximate computation of twin-width has attracted significant attention already since the moment the parameter was introduced. A recently proposed approach (STACS 2024) towards obtaining a better understanding of this question is to consider the approximability of twin-width via fixed-parameter algorithms whose running time depends not on twin-width itself, but rather on parameters which impose stronger restrictions on the input graph. The first step that article made in this direction is to establish the fixed-parameter approximability of twin-width (with an additive error of 1) when the runtime parameter is the feedback edge number.

Here, we make several new steps in this research direction and obtain:

- An asymptotically tight bound between twin-width and the feedback edge number;
- A significantly improved fixed-parameter approximation algorithm for twin-width under the same runtime parameter (i.e., the feedback edge number) which circumvents many of the technicalities of the original result and simultaneously avoids its formerly non-elementary runtime dependency;
- An entirely new fixed-parameter approximation algorithm for twin-width when the runtime parameter is the vertex integrity of the graph.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases twin-width, fixed-parameter algorithms, feedback edge number, vertex integrity

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.3

Related Version *Previous Version*: <https://arxiv.org/abs/2407.15514>

Funding *Robert Ganian*: Robert Ganian acknowledges support by the FWF and WWTF Science Funds (FWF project 10.55776/Y1329 and WWTF project ICT22-029).

Mathis Rocton: Mathis Rocton acknowledges support by the European Union’s Horizon 2020 research and innovation COFUND programme (LogiCS@TUWien, grant agreement No 101034440), and the FWF Science Fund (FWF project Y1329).



1 Introduction

Twin-width is a comparatively recent graph-theoretic measure which is the culmination of as well as a catalyst for several recent breakthroughs in the area of algorithmic model theory [10, 11, 12, 13, 14]. Indeed, it has the potential to provide a unified explanation of why model-checking first order logic is fixed-parameter tractable on a number of graph classes which were, up to then, considered to be separate islands of tractability for the model-checking problem. This includes graphs of bounded rank-width, proper minor-closed graphs, map graphs [15], bounded-width posets [3] as well as a number of other specialized graph classes [4, 22].



© Jakub Balabán, Robert Ganian, and Mathis Rocton;
licensed under Creative Commons License CC-BY 4.0

19th International Symposium on Parameterized and Exact Computation (IPEC 2024).

Editors: Édouard Bonnet and Paweł Rzażewski; Article No. 3; pp. 3:1–3:22

Leibniz International Proceedings in Informatics



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

While twin-width is related to graph parameters such as rank-width and path-width [14] as well as to measures which occur in matrix theory such as excluding linear minors [13], what distinguishes twin-width from these other measures is that we lack efficient algorithms for computing the twin-width of a graph. In particular, it is known that already deciding whether a graph has twin-width at most 4 is NP-hard [6]. This is highly problematic for the following reason: virtually every known algorithm that uses twin-width requires access to a so-called *contraction sequence*, which serves the same role as the decompositions typically used for classical parameters such as treewidth [36] and rank-width [35]. Intuitively speaking, a contraction sequence of width t – which serves as a witness for G having twin-width at most t – of a graph G is a sequence C of contractions of (not necessarily pairwise adjacent) vertex pairs which satisfies the following property: at each step of C , every vertex v only has at most t neighbors with an ancestor that is not adjacent to some ancestor of v ¹.

The aforementioned NP-hardness of identifying graphs of twin-width 4 [6] effectively rules out fixed-parameter as well as XP algorithms for computing optimal contraction sequences when parameterized by the twin-width itself. One possible approach to circumvent this obstacle would be to devise a fixed-parameter algorithm which still uses the twin-width t as the parameter and computes at least an approximately-optimal contraction sequences, i.e., a contraction sequence of width $f(t)$ for some computable function f . On a complexity-theoretic level, such a result may be seen as “almost” as good as computing twin-width exactly, as it would still yield a fixed-parameter algorithm for first-order model checking.

Unfortunately, the task of finding such an algorithm has proven to be highly elusive, and it is far from clear that one even exists – in fact, whether twin-width can be approximated in fixed-parameter time (for any function f of the twin-width) can be seen as arguably the most prominent open question in contemporary research of twin-width. Recently [2], we attacked this question by first relaxing the running time requirement and ask whether we can obtain an $f(t)$ -approximation for twin-width at least via a fixed-parameter algorithm where the runtime parameter is different (and, in particular, larger) than the twin-width t itself. As a first step in this direction, we developed a non-trivial fixed-parameter algorithm that computes a contraction sequence of width at most $t + 1$ and is parameterized by the *feedback edge number* of the input graph, i.e., the edge deletion distance to acyclicity [2]. In the same paper, we also showed that the twin-width of a graph with feedback edge number k is upper-bounded by $k + 1$.

Contributions. In this article, we significantly expand on our previous results [2] and present the next steps in the overarching program of understanding the boundaries of tractability for computing approximately-optimal contraction sequences. We summarize the three main contributions of this article below.

In Section 3, we revisit the relationship between twin-width and the feedback edge number. Here, we improve our previous linear bound [2] to square-root, and also show that this new bound is asymptotically tight. More precisely, we show that every graph class with feedback edge number k has twin-width $\mathcal{O}(\sqrt{k})$ (Theorem 8), and also construct a graph class with feedback edge number k whose twin-width is lower-bounded by $\Theta(\sqrt{k})$ (Proposition 9).

In Section 4, we revisit the main result of the preceding paper [2]: a polynomial-time reduction procedure which transforms every input graph G with feedback edge number k and twin-width t into a (tri-)graph G' whose twin-width lies between t and $t + 1$ and whose size is upper-bounded by a non-elementary function of k . While this suffices to obtain the

¹ Formal definitions are provided in Section 2.

desired fixed-parameter approximation algorithm (as one may brute-force over all contraction sequences of G'), the dependence on the parameter k is astronomical and the proof relies on a sequence of highly technical arguments about how a hypothetical contraction sequence may be retrofitted in order to avoid certain degenerate steps. As the second main contribution of this article, we provide a new proof for the fixed-parameter approximability of twin-width parameterized by the feedback edge number which not only avoids many of the technical difficulties faced in the previous approach, but crucially also improves the size bound for the reduced instance G' from a *non-elementary* to a *quadratic* function of k .

Finally, in Section 5 we push the frontiers of approximability for twin-width by obtaining an algorithm which computes a contraction sequence for G of width at most twice the graph's twin-width and runs in time $f(p) \cdot |G|$, where p is the *vertex integrity* of G . Vertex integrity is a parameter which intuitively measures how easily a graph may be separated into small parts, and is defined as the smallest integer p such that there exists a separator X with the following property: each connected component C of $G - X$ satisfies $|V(C) \cup X| \leq p$. Vertex integrity may be seen as the natural intermediate step between the *vertex cover number* (which is the size of the smallest vertex cover in G , and which is known to allow for a trivial fixed-parameter algorithm for computing twin-width) and decompositional parameters such as *treedepth* and *treewidth* (for which the existence of a fixed-parameter approximation algorithm for twin-width remains a prominent open question [2]). Our result relies on a data reduction procedure which incorporates entirely different arguments than those used for the feedback edge number, and the correctness proof essentially shows that every optimal contraction sequence can be transformed into a near-optimal one where all “similar parts” of G are treated in a “similar way”.

Related Work. Beyond the setting of computing twin-width and the associated contraction sequences, there are numerous other works which have targeted fixed-parameter algorithms for computing a structural graph parameter X when parameterized by graph parameters that differ from X . The general aim in this research direction is typically to further one's understanding of the fundamental problem of computing the targeted parameter X . Examples of fixed-parameter algorithms obtained in this setting include those for treewidth parameterized by the feedback vertex number [9], treedepth parameterized by the vertex cover number [33], MIM-width parameterized by the feedback edge number and other parameters [21], and the directed feedback vertex number parameterized by the (undirected) feedback vertex number [7]. The feedback edge number and vertex integrity have also been used to obtain parameterized algorithms for a number of other challenging problems [37, 5, 27, 34, 25, 29, 23, 30], whereas the latter parameter has also been studied in the literature under different asymptotically-equivalent names such as the *fracture number* [20, 24] and *starwidth* [38]. We refer interested readers to the very recent manuscript of Hanaka, Lampis, Vasilakis and Yoshiwatari [31] for a more detailed overview of vertex integrity and its relationship to other fundamental graph measures.

2 Preliminaries

For integers i and j , we let $[i, j] := \{n \in \mathbb{N} \mid i \leq n \leq j\}$ and $[i] := [1, i]$. We assume familiarity with basic concepts in graph theory [17] and parameterized algorithmics [18, 16]. When H is an induced subgraph of G , we denote it by $H \subseteq G$. Given vertex sets X and U , we will use $G[X]$ to denote the graph induced on X and $G - U$ to denote the graph $G[V(G) \setminus U]$; similarly, for an edge set F , $G - F$ denotes G after removing the edges in F .

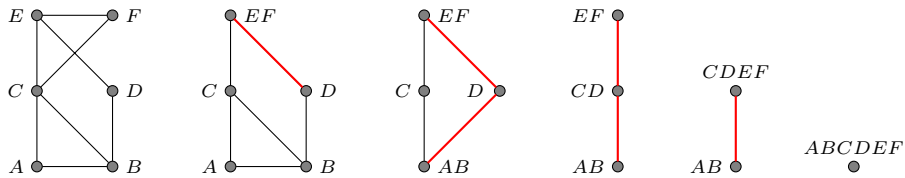
A *dangling path* in G is a path of vertices which all have degree 2 in G , and a *dangling tree* in G is an induced subtree in G which can be separated from the rest of G by removing a single edge. The *length* of a path is the number of edges it contains. The *distance* between two vertices u and v is the length of the shortest path between them.

An edge set F in an n -vertex graph G is called a *feedback edge set* if $G - F$ is acyclic, and the *feedback edge number* of G is the size of a minimum feedback edge set in G . We remark that a minimum feedback edge set can be computed in time $\mathcal{O}(n)$ as an immediate corollary of the classical (DFS- and BFS-based) algorithms for computing a spanning tree in an unweighted graph G .

A graph has *vertex integrity* p if p is the smallest integer with the following property: G contains a vertex set S such that $S \neq V(G)$ and for each connected component H of $G - S$, $|V(H) \cup S| \leq p$. One may observe that the vertex integrity is upper-bounded by the size of a minimum vertex cover in the graph (i.e., the vertex cover number) plus one, and both vertex integrity and the feedback edge number are lower-bounded by treewidth minus one [36]. The vertex integrity of an n -vertex graph can be computed in time $\mathcal{O}(p^{p+1} \cdot n)$ [19].

Twin-Width. A *trigraph* G is a graph whose edge set is partitioned into a set of *black* and *red* edges. The set of red edges is denoted $R(G)$, and the set of black edges $E(G)$. The *black* (resp. *red*) *degree* of $u \in V(G)$ is the number of black (resp. red) edges incident to u in G . We extend graph-theoretic terminology to trigraphs by ignoring the colors of edges; for example, the degree of u in G is the sum of its black and red degrees (in the literature, this is sometimes called the *total degree*). We say a (sub)graph is *black* (resp. *red*) if all of its edges are black (resp. red); for example, P is a red path in G if it is a path containing only red edges. Without a color adjective, the path (or a different kind of subgraph) may contain edges of both colors.

Given a trigraph G , a *contraction* of two distinct vertices $u, v \in V(G)$ is the operation which produces a new trigraph by (1) removing u, v and adding a new vertex w , (2) adding a black edge wx for each $x \in V(G)$ such that $xu, xv \in E(G)$, and (3) adding a red edge wy for each $y \in V(G)$ such that $yu \in R(G)$, or $yv \in R(G)$, or y contains only a single black edge to either v or u . A sequence $C = (G = G_1, \dots, G_n)$ is a *partial contraction sequence* of G if it is a sequence of trigraphs such that for all $i \in [n - 1]$, G_{i+1} is obtained from G_i by contracting two vertices. A *contraction sequence* is a partial contraction sequence which ends with a single-vertex graph. The *width* of a (partial) contraction sequence C , denoted $w(C)$, is the maximum red degree over all vertices in all trigraphs in C . The *twin-width* of G , denoted $\text{tw}(G)$, is the minimum width of any contraction sequence of G , and a contraction sequence of width $\text{tw}(G)$ is called *optimal*. An example of a contraction sequence is provided in Figure 1.



■ **Figure 1** A contraction sequence of width 2 for the leftmost graph, consisting of 6 trigraphs.

Let us now fix a contraction sequence $C = (G = G_1, \dots, G_n)$. For each $i \in [n]$, we associate each vertex $u \in V(G_i)$ with a set $\beta(u, i) \subseteq V(G)$, called the *bag* of u , which contains all vertices contracted into u . Formally, we define the bags as follows:

- for each $u \in V(G)$, $\beta(u, 1) := \{u\}$;
- for $i \in [n - 1]$, if w is the new vertex in G_{i+1} obtained by contracting u and v , then $\beta(w, i + 1) := \beta(u, i) \cup \beta(v, i)$; otherwise, $\beta(w, i + 1) := \beta(w, i)$.

Note that if a vertex u appears in multiple trigraphs in C , then its bag is the same in all of them, and so we may denote the bag of u simply by $\beta(u)$. Let us fix $i, j \in [n]$, $i \leq j$. If $u \in V(G_i)$, $v \in V(G_j)$, and $\beta(u) \subseteq \beta(v)$, then we say that u is an *ancestor* of v in G_i and v is the *descendant* of u in G_j (clearly, this descendant is unique). If H is an induced subtrigraph of G_i , then $u \in V(G_j)$ is a *descendant* of H if it is a descendant of at least one vertex of H . A contraction of $u, v \in V(G_j)$ into $w \in V(G_{j+1})$ *involves* $w \in V(G_i)$ if w is an ancestor of uv .

The following definition provides terminology that allows us to partition a contraction sequence into “steps” based on contractions between a subset of vertices in the original graph.

► **Definition 1.** Let C be a contraction sequence of a trigraph G , and let H be an induced subtrigraph of G with $|V(H)| = m$. For $i \in [m - 1]$, let $C\langle i \rangle_H$ be the trigraph in C obtained by the i -th contraction between two descendants of H , and let $C\langle 0 \rangle_H = G$. For $i \in [m - 1]$, let u_i and w_i be the two vertices that are contracted into the new vertex of $C\langle i \rangle_H$.

A contraction sequence $C[H] = (H = H_1, \dots, H_m)$ is the restriction of C to H if for each $i \in [m - 1]$, H_{i+1} is obtained from H_i by contracting the two vertices $u, w \in V(H_i)$ such that $\beta(u) = \beta(u_i) \cap V(H)$ and $\beta(w) = \beta(w_i) \cap V(H)$.

It will also be useful to have an operation that forms the “reverse” of a restriction; we define this below.

► **Definition 2.** Let G and H be graphs such that $H \subseteq G$ and let C_0 be a partial contraction sequence of H . We say that a partial contraction sequence C of G is the extension of C_0 to G if $C[H] = C_0$ and no contraction in C involves a vertex of $G - H$. When G_i is the i -th trigraph in C_0 , we denote by $G_i \uparrow G$ the i -th trigraph in C (this makes sense since the lengths of C_0 and C are the same).

Finally, we introduce a notion that will be useful when dealing with reduction rules in the context of computing contraction sequences.

► **Definition 3.** Let G, G' be trigraphs. We say that the twin-width of G' is effectively at most the twin-width of G , denoted $\text{tw}(G') \leq_e \text{tw}(G)$, if (1) $\text{tw}(G') \leq \text{tw}(G)$ and (2) given a contraction sequence C of G , a contraction sequence C' of G' of width at most $w(C)$ can be constructed in polynomial time. If $\text{tw}(G') \leq_e \text{tw}(G)$ and $\text{tw}(G) \leq_e \text{tw}(G')$, then we say that the two graphs have effectively the same twin-width, $\text{tw}(G') =_e \text{tw}(G)$.

Preliminary Observations and Remarks. We begin by stating a simple brute-force algorithm for computing twin-width.

► **Observation 4.** An optimal contraction sequence of an n -vertex graph can be computed in time $2^{\mathcal{O}(n \cdot \log n)}$.

Proof. Each contraction sequence is defined by $n - 1$ choices of a pair of vertices, and so the number of contraction sequences is $\mathcal{O}((n^2)^n) = \mathcal{O}(2^{2n \cdot \log n}) \leq 2^{\mathcal{O}(n \cdot \log n)}$. Moreover, computing the width of a contraction sequence can clearly be done in polynomial time. ◀

The following observation provides a useful insight into the optimal contraction sequences of trees.

► **Observation 5** ([15, Section 3]). *For any rooted tree T with root r , there is a contraction sequence C of T of width at most 2 such that the only contraction involving r is the very last contraction in C .*

3 The Square-Root Bound

In this section, we prove that a graph with feedback edge number k has twin-width at most $\mathcal{O}(\sqrt{k})$. On a high level, the idea we will employ here builds on the preprocessing techniques originally introduced in the context of computing twin-width on tree-like graphs [2]: first we will contract the dangling trees, then the dangling paths, and for the final step we will use the following theorem of Ahn, Hendrey, Kim and Oum:

► **Theorem 6** ([1]). *If G is a graph with m edges, then the twin-width of G is at most $\sqrt{3m} + o(\sqrt{m})$.*

An issue we need to resolve before applying the aforementioned high-level approach is that Theorem 6 only applies to graphs without red edges, whereas the trigraph G we will obtain after dealing with the dangling trees and paths may contain these. The following lemma shows, using the properties of the red edges in G , that making all edges of G black can only decrease the twin-width by a constant.

► **Lemma 7.** *Let G be a trigraph with maximum red degree 2 such that each red edge in G is incident to a vertex of degree at most 2. If G' is the graph obtained from G by making all edges black, then $\text{tww}(G) \leq \text{tww}(G') + 4$.*

Proof. Let C' be an optimal contraction sequence of G' , and let C be the contraction sequence of G obtained by following C' . We will prove that $w(C) \leq w(C') + 4$.

Let G_i be any trigraph in C and let G'_i be the trigraph in C' such that $V(G_i) = V(G'_i)$. Suppose for a contradiction that there are distinct vertices $u, v_1, v_2, v_3, v_4, v_5 \in V(G_i)$ such that for each $j \in [5]$, uv_j is a red edge in G_i but not in G'_i . Recall that $\beta(w)$ denotes the set of vertices contracted to w (the bag of w), and observe that for each $j \in [5]$, there must be vertices $u_j, v_j^0 \in V(G)$ such that $u_j \in \beta(u)$ and $v_j^0 \in \beta(v_j)$, and $u_j v_j^0$ is an edge that is black in G' but red in G . Since $uv_j \notin R(G'_i)$, there must be either all edges or no edges between $\beta(u)$ and $\beta(v_j)$ in G' . However, $u_j v_j^0 \in E(G')$, which means that for all $j, \ell \in [5]$, $u_j v_\ell^0$ is a black edge in G' (and so it is an edge also in G).

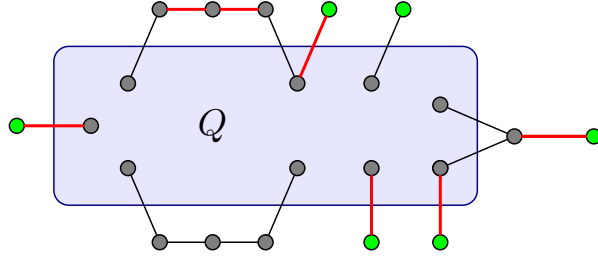
Since all vertices of G have red degree at most 2 and $u_j v_j^0 \in R(G)$ for each $j \in [5]$, there must be $a, b, c \in [5]$ such that $|\{u_a, u_b, u_c\}| = 3$. Now observe that each vertex in $\{u_a, u_b, u_c, v_a^0, v_b^0, v_c^0\}$ has degree at least 3 in G (since $u_j v_\ell^0$ is an edge in G for all $j, \ell \in \{a, b, c\}$). However, each red edge in G has an endpoint of degree at most 2, which is a contradiction.

We have proven that the red degree of each vertex $u \in V(G_i) = V(G'_i)$ may be higher in G_i than in G'_i by at most 4, which proves that $w(C) \leq w(C') + 4$. ◀

We are now ready to prove the square-root upper bound on twin-width.

► **Theorem 8.** *There exists a function $f(k) \in \mathcal{O}(\sqrt{k})$ such that every graph G with feedback edge number k has twin-width at most $f(k)$.*

Proof. Let F be a smallest feedback edge set of G and assume $k = |F| > 0$ (the case $k = 0$ follows from Observation 5). We will prove the statement by constructing a contraction sequence for G of width at most $f(k) \in \mathcal{O}(\sqrt{k})$. We begin by contracting each maximal dangling tree to a single vertex using Observation 5. After a maximal dangling tree has been



■ **Figure 2** A trigraph after processing the dangling trees and shortening the paths in \mathcal{P} . Spikes are colored in green. The edges between vertices of Q are not depicted. Notice that one of the paths is black: this means it had no spikes and it has not been shortened. Also notice that one spike is attached by a black edge: it was a maximal dangling tree with only one vertex in G .

contracted, we call the last remaining vertex a *spike*, and we say that a vertex adjacent to a spike *has a spike*. Whenever a vertex has two spikes, we contract the spikes together (the obtained vertex is still called a spike). Observe that throughout this process, no vertex has red degree higher than 2: this is ensured by Observation 5 and the fact that a red neighbor of a vertex not in a dangling tree must be a spike.

Let G^α be the obtained trigraph and let T be the tree obtained from G^α by removing all spikes and edges in F . Let us choose any vertex of T to be the root. Let $Q_0 := \{u \in V(T) \mid u \text{ is incident to an edge of } F \text{ in } G^\alpha\}$, $Q_1 := \{u \in V(T) \mid u \text{ has degree higher than 2 in } T\}$, and $Q := Q_0 \cup Q_1$. It is easy to see that $|Q_0| \leq 2k$ and that all leaves of T belong to Q_0 (a leaf not in Q_0 would belong to a dangling tree in G^α). Since a tree with n leaves has at most n vertices of degree higher than 2, we obtain that $|Q_1| \leq 2k$ and $|Q| \leq 4k$. Observe that $T - Q$ is a graph consisting of disjoint dangling paths. Let \mathcal{P} be the set of these paths, and let $g: \mathcal{P} \rightarrow Q$ be the function such that $g(P)$ is the vertex of Q adjacent to the endpoint of P that is farther from the root of T . Since g is injective, we obtain that $|\mathcal{P}| \leq 4k$.

For each path $P = (u_1, \dots, u_n)$ in \mathcal{P} , we perform the following contractions (starting with G^α).

- If $n > 2$, then for each $i \in [2, n-1]$ such that u_i has a spike v , contract u_i and v (do this in increasing order). If u_1 (resp. u_n) has a spike v , contract v and u_2 (resp. u_{n-1}).
- If $n > 3$, shorten P to a path with exactly three vertices by repeatedly contracting neighboring vertices of $P - \{u_1, u_n\}$.

Observe that throughout this process, no vertex has red degree higher than 2: a vertex in Q has red degree at most 1 (its red neighbor must be a spike) and a vertex in a path P either has a spike and at most one red neighbor in P or at most two red neighbors in P . See Figure 2 for an illustration.

Now we will count the number of edges in the obtained trigraph G^β . First, observe that there are at most $5k$ edges in $G^\beta[Q]$: k edges belonging to F and at most $4k$ other edges since $G^\beta[Q] - F$ is a forest. In addition, each vertex of Q may have a spike in G^β , which constitutes up to $4k$ other edges. Second, let $P \in \mathcal{P}$. If the length of P in G is at least 2, then P corresponds to at most four edges in G^β : at most two edges of the path itself and two edges connecting P to the rest of the graph (i.e., to vertices of Q). However, if P is shorter in G , then its vertices may have spikes in G^β and it may correspond to up to 5 edges: one edge of the path, two edges connecting it to Q , and two edges going to the spikes. Hence, \mathcal{P} adds at most $20k$ edges, and thus there are at most $29k$ edges in G^β .

Let G^γ be the graph obtained from G^β by changing the color of all edges to black. By Theorem 6, G^γ has twin-width at most $\sqrt{87k} + o(\sqrt{k})$. Notice that G^β satisfies the preconditions of Lemma 7, which means that $\text{tww}(G^\beta) \leq \text{tww}(G^\gamma) + 4$. Hence, the twin-width of G^β is also at most $\sqrt{87k} + o(\sqrt{k})$, and the same also holds for the original graph G (since the partial contraction sequence from G to G^β has width at most 2). \blacktriangleleft

We conclude the section by showing that Theorem 8 is asymptotically tight.

► **Proposition 9.** *There exists a function $f(k) \in \Omega(\sqrt{k})$ and an infinite class \mathcal{G} of graphs such that for each $G \in \mathcal{G}$ with feedback edge number k , $\text{tww}(G) \geq f(k)$.*

Proof. Let n be a prime power such that $n \equiv 1 \pmod{4}$. It is known that there exists an n -vertex $((n-1)/2)$ -regular graph G (a so-called Paley graph) that has twin-width exactly $(n-1)/2$ [1, Section 3]. Since $|E(G)| = (n^2 - n)/4$ and the spanning forest of G has at most $n-1$ edges, we know that the feedback edge number k of G is at least $(n^2 - 5n + 4)/4 \in \Omega(n^2)$.

Let \mathcal{G} be the class of all such n -vertex Paley graphs. For each n -vertex graph G in \mathcal{G} , we have $k \in \Omega(n^2)$ and $\text{tww}(G) \in \Theta(n)$. Let f be the function which maps each k to the minimum of $\{\text{tww}(G) \mid G \in \mathcal{G} \text{ is a graph with feedback edge number } k\}$. Thus, for each $G \in \mathcal{G}$, $\text{tww}(G) \geq f(k)$, and the aforementioned relationships between the number n of vertices of that graph, $\text{tww}(G)$ and k guarantee that $f(k) \in \Omega(\sqrt{k})$, as desired. \blacktriangleleft

4 A Better Algorithm Parameterized by the Feedback Edge Number

We begin by recalling that the case of twin-width 2 is known to already admit an exact nearly single-exponential fixed-parameter algorithm parameterized by the feedback edge number (see Theorem 10 below), and thus here we focus our efforts on graphs with higher twin-width.

► **Theorem 10** ([2]). *If G is a graph with feedback edge number k and $\text{tww}(G) \leq 2$, then an optimal contraction sequence of G can be computed in time $2^{\mathcal{O}(k \cdot \log k)} + n^{\mathcal{O}(1)}$.*

Our algorithm uses the same initial preprocessing steps as our previous result [2]. These are formalized through the following definition and theorem; note that in the approach we use here, we can use a slightly more general (and less technical) definition of *tidy* (H, \mathcal{P}) -graphs than the preceding paper.

► **Definition 11.** *A connected trigraph G with $\text{tww}(G) \geq 2$ is a tidy (H, \mathcal{P}) -graph if \mathcal{P} is a non-empty set of dangling red paths in G , and there are two disjoint induced subtrigraphs of G , namely H and $\sqcup \mathcal{P}$ (the disjoint union of all paths in \mathcal{P}), such that each vertex of G belongs to one of them. Moreover, if $u \in V(H)$ has a neighbor $v \in V(\sqcup \mathcal{P})$ in G , then u has black degree 0 in G , and v is the only neighbor of u in $\sqcup \mathcal{P}$.*

The following theorem summarizes the results obtained in [2] that we will use in this section.

► **Theorem 12** ([2], Theorem 17 + Corollary 20). *There is a polynomial-time procedure which takes as input a graph G with feedback edge number k and either outputs an optimal contraction sequence of G of width at most 2, or a tidy (H, \mathcal{P}) -graph G' with effectively the same twin-width as G such that $|V(H)| \leq 112k$ and $|\mathcal{P}| \leq 4k$.*

From here on, we pursue an entirely different approach than the one used to obtain the previous (non-elementary) kernel [2]. In Subsection 4.1, we show how a tidy (H, \mathcal{P}) -graph can be contracted when the paths in \mathcal{P} are long enough and a contraction sequence of H is given. This is then used in Subsection 4.2, where we describe a better algorithm for approximating twin-width parameterized by the feedback edge number (see Theorem 18).

4.1 Contracting an (H, \mathcal{P}) -Graph Using a Contraction Sequence for H

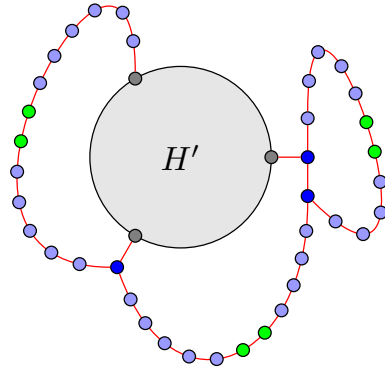
For this subsection, let us fix a tidy (H, \mathcal{P}) -graph G and let $m := |\mathcal{P}|$. Assume that each $P \in \mathcal{P}$ satisfies $|V(P)| \geq 8m$ and let F be the subtrigraph of $\sqcup \mathcal{P}$ induced by the vertices at distance at most $2m$ from H in G .

Informally speaking, our goal now is to construct a “good” contraction sequence for such a trigraph G , see Corollary 16. To achieve that, we need to describe some well-structured trigraphs obtained by a sequence of contractions from G , which we will call G -tidy trigraphs, see the following Definition 13. An important property of a G -tidy trigraph is that all contractions happened either between two vertices of H or two vertices of F at the same distance from H (see items 1 and 3).

► **Definition 13.** Let G' be a trigraph obtained by a sequence of contractions from G and let H' (resp. F') be the subtrigraph of G' induced by the vertices u such that $\beta(u)$ is a subset of $V(H)$ (resp. $V(F)$). We say that G' is a G -tidy trigraph if:

1. For $u \in V(G')$, we have $u \in V(H') \cup V(F')$ or $|\beta(u)| = 1$.
2. Each $u \in V(H')$ has at most one neighbor outside of H' in G' .
3. For each $u \in V(F')$, all vertices in $\beta(u)$ have the same distance d from H in G . We say that d is the level of u .
4. F' is a forest such that all its vertices have degree at most 3 in G' . If T is a connected component of F' , then:
 - a. T has exactly one vertex r at level 1 (let us declare it the root of T).
 - b. The vertices of T with degree 3 in G_i form a subtree T' of T .
 - c. The vertices of T' have level at most $|\beta(r)| - 1$, and either $T' = \emptyset$ or $r \in V(T')$.

See Figure 3 for an illustration.



► **Figure 3** An illustration of Definition 13 when $m = 3$. The depicted G -tidy trigraph G' consists of H' : vertices colored in grey, F' : vertices colored in blue (degree-3 vertices in darker shade), and the remaining vertices are colored in green. The edges inside of H' are not depicted (there can be both red and black edges). Note that instead of each pair of green vertices, there should be at least 12 of them (because each path in \mathcal{P} should contain at least $8m = 24$ vertices).

Now we show how G can be reduced to a G -tidy trigraph with H' being a single-vertex graph; this will be the first part of the proof of Corollary 16. Note that the assumption that C_H is given will be later handled in the proof of Lemma 17.

► **Lemma 14.** Given a contraction sequence C_H of H , one can compute a partial contraction sequence of width $\max(w(C_H) + 1, 4)$ from G to a G -tidy trigraph G' with $|V(H')| = 1$, in polynomial time.

Proof. For each $i \in [|V(H)|]$, we will construct a partial contraction sequence C_i from G to a G -tidy trigraph G_i such that $w(C_i) \leq \max(w(C_H) + 1, 4)$ and the restriction of C_i to H will be the prefix of C_H of length i . We will denote the subtrigraphs of G_i corresponding to H' and F' (see Definition 13) by H_i and F_i , respectively. We define $C_1 = (G)$, i.e., C_1 is the trivial partial contraction sequence with no contractions. It can be easily verified that $G_1 := G$ is a G -tidy trigraph. In particular, the forest $F_1 := F$ consists of $2m$ disjoint paths.

Suppose that we have constructed C_i for some $i < |V(H)|$. Let $u, v \in V(H_i)$ be the two vertices contracted in H_{i+1} (which is the successor of H_i in C_H). If u or v does not have a neighbor outside of H_i in G_i , then we define G_{i+1} to be the trigraph obtained from G_i by contracting u and v . Clearly, G_{i+1} is a G -tidy trigraph and C_{i+1} (the sequence obtained by prolonging C_i with G_{i+1}) has the required properties. Now suppose that both u and v have a neighbor outside of H_i in G_i . In this case, we cannot simply contract them because the new vertex would have two neighbors outside of H_{i+1} , violating condition 2 of Definition 13.

Let T_u and T_v be the two connected components of F_i with roots adjacent to u and v , respectively. Informally, we need to merge T_u and T_v before we can contract u and v . Let $T \in \{T_u, T_v\}$ be a tree with root r . If T contains no degree-3 vertices, we do nothing (we always mean degree in G_i). Otherwise, let $w \in V(T)$ be the deepest degree-3 vertex such that all its ancestors in T have degree 3. By item 3 of Definition 13, $|\beta(r)| \leq 2m$ because F contains exactly $2m$ vertices at level 1 (2 for each path in \mathcal{P}). Hence, by item 4c, the level of w is less than $2m$, and so w has two children x and y in T , both of degree 2. We contract x and y (note that the obtained vertex xy has red degree 3, and the red degree of w drops to 2). We repeat this process as long as such vertex w exists (crucially, xy cannot be chosen as the next w because its parent has degree 2). Afterwards, we contract the roots r_u, r_v of T_u and T_v , and finally, we contract u and v into uv .

Let C_{i+1} be the partial contraction sequence of G obtained by prolonging C_i with the contractions described in the previous paragraph. Let us show that $w(C_{i+1}) \leq \max(w(C_H) + 1, 4)$. By the assumption about C_i , it suffices to discuss red degrees in each trigraph G' between G_i and G_{i+1} (which is the last trigraph in C_{i+1}). Clearly, any descendant of H in G' has red degree at most $w(C_H) + 1$ (it is crucial that u and v are contracted after r_u and r_v). Any other vertex of G' has red degree at most 3, except for the vertex obtained by contracting r_u and r_v , whose red degree is 4 (but it drops to 3 when u and v are contracted).

Finally, we need to show that G_{i+1} is G -tidy. It is easy to see that G_{i+1} satisfies the first three items of Definition 13. To prove that G_{i+1} satisfies item 4, observe that F_{i+1} is indeed a forest: it contains the same trees as F_i , except that T_u and T_v have been merged into a new tree T with root r . More precisely, T is isomorphic to the tree obtained from the disjoint union of T_u and T_v by first adding a new vertex r and edges rr_u, rr_v , and second removing all leaves. Since r has degree 3 in G_{i+1} , the highest level of a degree-3 vertex in T is one higher than in the union of T_u and T_v in G_i . Since $|\beta(r)| = |\beta(r_u)| + |\beta(r_v)|$ and both of these summands are at least 1, we get that G_{i+1} satisfies item 4c, which concludes the proof. \blacktriangleleft

To prove Corollary 16, we now show that the G -tidy trigraph given by Lemma 14 can be contracted to a single vertex (without creating vertices with high red degree). Note that the following proof is inspired by the proof of Theorem 7 in [6].

► **Lemma 15.** *If G' is a G -tidy trigraph G' with $|V(H')| = 1$, then a contraction sequence of G' of width at most 4 can be computed in polynomial time.*

Proof. First, observe that by Definition 11, all edges in G' are red. By item 2 of Definition 13, the only vertex u of H' has a single neighbor r . By Definition 11, G is connected; hence, also G' is connected. This implies that F' is a tree. Let T be the subtree of F' induced by the vertices with degree 3 in G' . Let us begin by contracting u and r , obtaining a trigraph $G^* := G' - u$.

Observe that the depth of T is at most $2m - 1$ because r contains $2m$ vertices in its bag (by property 4c). Consider a path $P \in \mathcal{P}$ and observe that the descendants of at most $4m$ vertices of P belong to T in G^* ($2m$ from each side). Hence, $G^* - T$ consists of disjoint dangling red paths, each with at least $4m$ vertices (since each $P \in \mathcal{P}$ satisfies $|V(P)| \geq 8m$). Let \mathcal{P}' be the set of these red paths in G^* .

Let $P \in \mathcal{P}'$ and let u and u' be the endpoints of P . Let $v, v' \in V(T)$ be the neighbors of u and u' in T , respectively, and let Q be the path connecting v and v' in T . Since the depth of T is at most $2m - 1$, we know that Q contains at most $4m - 1$ vertices. Let us shorten P so that it has the same length as Q (by repeatedly contracting consecutive vertices). Let $(u_1 = u, \dots, u_p = u')$ and $(v_1 = v, \dots, v_p = v')$ be the sequences of vertices of P and Q in the natural orders. Now for each $i \in [p]$ in increasing order, contract u_i and v_i , and observe that the obtained trigraph is isomorphic to $G^* - P$. Repeat this for all paths $P \in \mathcal{P}'$, obtaining a trigraph isomorphic to T , which has twin-width at most 3 and can be contracted as per Observation 5. Finally, observe that during a contraction of a path in $P \in \mathcal{P}'$, there is never a vertex with red degree higher than 4. Indeed, after contracting u_i and v_i for $i \in [p - 1]$, the obtained vertex has at most four red neighbors: at most three in T plus u_{i+1} . ◀

4.2 Wrapping up the Proof

In the previous subsection, we proved Lemmas 14 and 15, which together imply the following corollary.

► **Corollary 16.** *Let G be a tidy (H, \mathcal{P}) -graph such that each $P \in \mathcal{P}$ satisfies $|V(P)| \geq 8 \cdot |\mathcal{P}|$. Given a contraction sequence C_H of H , one can compute a contraction sequence of G of width $\max(w(C_H) + 1, 4)$, in polynomial time.*

Now we are able to show that if we shorten all long paths in a tidy (H, \mathcal{P}) -graph, then the twin-width increases by at most 1 (formally, shortening a path means contracting its consecutive vertices).

► **Lemma 17.** *Let G be a tidy (H_0, \mathcal{P}_0) -graph such that $\text{tw}(G) \geq 3$, let $m = |\mathcal{P}_0|$ and let G' be the trigraph obtained from G by shortening each path $P \in \mathcal{P}_0$ with more than $8 \cdot m$ vertices to length exactly $8 \cdot m - 1$. Then $\text{tw}(G') \leq \text{tw}(G) + 1$.*

Proof. We begin by handling short paths in \mathcal{P}_0 : let $\mathcal{P}_{\text{short}} = \{P \in \mathcal{P}_0 : |V(P)| < 8m\}$, let H be the union of H_0 and $\sqcup \mathcal{P}_{\text{short}}$ (including the edges between them), and let $\mathcal{P} = \mathcal{P}_0 \setminus \mathcal{P}_{\text{short}}$. Clearly, G is also a tidy (H, \mathcal{P}) -graph. Also observe that G' is a tidy (H, \mathcal{P}') -graph (where \mathcal{P}' is the set of paths obtained from \mathcal{P} by shortening each path in it).

We want to construct a contraction sequence C' of G' of width at most $\text{tw}(G) + 1$ from an optimal contraction sequence C of G . Let C_H be the restriction of C to H ; clearly, $w(C_H) \leq \text{tw}(G)$. Since $\text{tw}(G) \geq 3$, it suffices to apply Corollary 16 on G' using C_H , which yields the desired contraction sequence C' . ◀

Finally, we are able to prove the main result of this section.

► **Theorem 18.** *Given a graph G with feedback edge number k , a trigraph G' of size $\mathcal{O}(k^2)$ such that $\text{tw}(G) \leq \text{tw}(G') \leq \text{tw}(G) + 1$ can be computed in polynomial time. Moreover, a contraction sequence for G of width at most $\text{tw}(G) + 1$ can be computed in time $2^{\mathcal{O}(k^2 \cdot \log k)} + n^{\mathcal{O}(1)}$.*

Proof. First, we use Theorem 10 to check whether $\text{tw}(G) \leq 2$ (if yes, G' can be any constant-size graph with the same twin-width as G). From now on, assume $\text{tw}(G) \geq 3$. Now let us use Theorem 12 to obtain a tidy (H, \mathcal{P}) -graph G_1 with effectively the same twin-width as G such that $|V(H)| \leq 112k$ and $|\mathcal{P}| \leq 4k$. Let G' be the trigraph obtained when Lemma 17 is applied on G_1 . By Lemma 17, $\text{tw}(G') \leq \text{tw}(G_1) + 1$. Conversely, $\text{tw}(G') \geq \text{tw}(G_1)$ because there is a partial contraction sequence C_1 from G_1 to G' of width at most $\text{tw}(G')$; it suffices to shorten paths of \mathcal{P} that are shorter in G' than in G_1 by contracting consecutive vertices. Hence, we indeed have $\text{tw}(G) \leq \text{tw}(G') \leq \text{tw}(G) + 1$.

Next, let us examine the size of G' . By Lemma 17, each of the $4k$ paths in \mathcal{P} has at most $8 \cdot 4k$ vertices in G' . Hence, we obtain $|V(G')| \leq 128k^2 + 112k \in \mathcal{O}(k^2)$ as required.

Finally, let us show how a contraction sequence for G of width at most $\text{tw}(G) + 1$ can be computed. If $\text{tw}(G) \leq 2$, then this contraction sequence is provided by Theorem 10. Otherwise, observe that an optimal contraction sequence C' of G' can be computed in time $2^{\mathcal{O}(k^2 \cdot \log k)}$ by Observation 4. Next we concatenate C' and C_1 (which is defined above and can be computed trivially) to obtain a contraction sequence of G_1 of width at most $\text{tw}(G_1) + 1$. We conclude using the effectiveness part of $\text{tw}(G) =_e \text{tw}(G_1)$ (see Definition 3). ◀

5 A Fixed-Parameter Algorithm Parameterized by Vertex Integrity

In this section, we design an FPT 2-approximation algorithm for computing twin-width when parameterized by the vertex integrity, see Theorem 23.

5.1 Initial Setup and Overview

For the following, it will be useful to recall the definition of vertex integrity presented in Section 2. Let us fix a graph G and a choice of $S \subseteq V(G)$ witnessing that the vertex integrity of an input graph G is p , and let \mathcal{C} be the set of connected components of $G - S$. We assume without loss of generality that G is connected, as the twin-width of a graph is the maximum twin-width of its connected components. We now define a notion of “component-types” which intuitively captures the equivalence between components which exhibit the same outside connections and internal structure.

► **Definition 19.** We say that two graphs $H_0, H_1 \in \mathcal{C}$ are twin-blocks, denoted $H_0 \sim H_1$, if there exist a canonical isomorphism α from H_0 to H_1 such that for each vertex $u \in V(H_0)$ and each $v \in S$, $uv \in E(G)$ if and only if $\alpha(u)v \in E(G)$. Clearly, \sim is an equivalence relation.

In a nutshell, our algorithm first computes an optimal contraction sequence C' for a subgraph G' of G that is obtained by keeping only a bounded number of twin-blocks from each equivalence class, and then uses C' to obtain a contraction sequence for G of width at most $2 \cdot \text{tw}(G') \leq 2 \cdot \text{tw}(G)$. In the following definition, we introduce terminology related to subgraphs of G .

► **Definition 20.** Let G' be an induced subgraph of G .

- We say that G' is \mathcal{C} -respecting if $S \subseteq V(G')$ and for each $H \in \mathcal{C}$, either $H \subseteq G'$ or $V(H) \cap V(G') = \emptyset$.
- We say that an equivalence class $[H_0]$ of \sim is large in G' if $|\mathcal{H}| \geq f(p)$, where $\mathcal{H} = \{H \in [H_0] \mid H \subseteq G'\}$ and $f(p) = 2^{7p^3}$.
- We say that G' is the reduced graph of G if it is obtained from G by removing all but $f(p)$ twin-blocks from each large class of \sim .

Let us now bound the size of the reduced graph G' .

► **Observation 21.** *If G' is the reduced graph of G , then $|V(G')| \leq p + p^2 \cdot f(p) \cdot 2^{2p^2} \in 2^{\mathcal{O}(p^3)}$.*

Proof. First, let us compute the size of \mathcal{C}/\sim . Each $H \in \mathcal{C}$ has at most p vertices, which means that the number of non-isomorphic graphs in \mathcal{C} can be upper-bounded by $p \cdot 2^{p^2}$. Since $|S| \leq p$, there are at most p^2 possible edges between S and each $H \in \mathcal{C}$. Hence, $|\mathcal{C}/\sim| \leq p \cdot 2^{2p^2}$. Because $|V(H)| \leq p$ for each $H \in \mathcal{C}$ and by definition of G' , the union of each class of \sim contains at most $p \cdot f(p)$ vertices. Finally, we again use that $|S| \leq p$. ◀

The core of our algorithm is the following lemma, which we will prove in Subsection 5.2:

► **Lemma 22.** *If G' is the reduced graph of G , then given a contraction sequence C' for G' of width t , we can compute a contraction sequence for G of width at most $2t$ in polynomial time.*

Let us now show how we can use this lemma to design the desired algorithm:

► **Theorem 23.** *If G is a graph with vertex integrity p , then a contraction sequence for G of width at most $2 \cdot \text{tww}(G)$ can be computed in time $g(p) \cdot n^{\mathcal{O}(1)}$, where g is an elementary function.*

Proof. The first step of the algorithm is to compute an optimal vertex-integrity decomposition of G . As noted already in Section 2, this can be done in time $\mathcal{O}(p^{p+1} \cdot n)$ [19]. Using this decomposition, we can compute the reduced graph G' of G in polynomial time. Next, we can compute an optimal contraction sequence C' of G' , using Observation 4. Since the size of G' is bounded (see Observation 21), we deduce that computing C' takes time $g(p) \in \exp(\exp(\mathcal{O}(p^3)))$, where $\exp(x) = 2^x$.

Finally, we apply Lemma 22 to compute in polynomial time a contraction sequence C for G of width at most $2 \cdot w(C') = 2 \cdot \text{tww}(G')$. Since G' is an induced subgraph of G , we know $\text{tww}(G') \leq \text{tww}(G)$, which implies the desired bound $w(C) \leq 2 \cdot \text{tww}(G)$. ◀

5.2 Extending a contraction sequence from G' to G

This subsection is dedicated to proving Lemma 22. Recall that we have fixed a graph G and a set $S \subseteq V(G)$, and that \mathcal{C} is the set of connected components of $G - S$. Let us begin with several technical definitions.

► **Definition 24.** *Let G' be a \mathcal{C} -respecting graph, let H_0 and H_1 be distinct twin-blocks (with canonical isomorphism α) such that $H_0, H_1 \subseteq G'$, and let G^* be any trigraph obtained from G' by a sequence of contractions. We say that H_0 and H_1 are merged in G^* if, for each $u \in V(H_0)$, there is a vertex $v \in V(G^*)$ such that $u, \alpha(u) \in \beta(v)$.*

It might be confusing that in the following definition, we consider a \mathcal{C} -respecting graph and a graph $H \in \mathcal{C}$ that is *not* its induced subgraph. The reason for this is that later we will show that, under some conditions, H can be “added” without increasing the twin-width too much. In fact, all such graphs H will be progressively added until all of them are present (and the obtained graph is the whole G). To formalize the process of adding H , we will use Definition 2 to create an extension of a contraction sequence to a sequence with H “appended” to all trigraphs.

► **Definition 25.** Let G' be a \mathcal{C} -respecting graph, let $H \in \mathcal{C}$ be such that $H \not\subseteq G'$, let $C' = (G'_1, G'_2, \dots)$ be a contraction sequence of G' .

- We say that a trigraph G'_i in C' is the C' -critical trigraph for H if i is the least index such that some vertex of H has a red neighbor in $G'_i \uparrow G$.
- If G'_i is the C' -critical trigraph for H , then we say that a trigraph G'_j is C' -safe for H if $j < i$ and there are two graphs $H', H'' \in [H]_{\sim}$ that are merged in G'_j .

We will show that for each H and C' (as in Definition 25), there is a C' -safe trigraph for H . The first step towards this is to show that if H has many twin-blocks in G' , then there are two twin-blocks of H merged in the C' -critical trigraph G^* for H . Intuitively, if the twin-blocks of H were not “sufficiently-merged” in G^* , then some vertex of S would have high red degree because the existence of a red edge between S and H (see the definition of C' -critical) implies red edges between S and all twin-blocks of H .

► **Lemma 26.** If G' is a \mathcal{C} -respecting graph, C' is a contraction sequence of G' , $H \in \mathcal{C}$ is a graph such that $H \not\subseteq G'$, the class $\mathcal{H} := [H]_{\sim}$ is large in G' , and G^* is the C' -critical trigraph for H , then there are two graphs $H', H'' \in [H]_{\sim}$ that are merged in G'_i .

Proof. Let $I = [f(p)]$ and let $H_1, \dots, H_{f(p)} \in \mathcal{H}$ be distinct graphs such that $H_i \subseteq G'$ for each $i \in I$ (using the fact that \mathcal{H} is large in G'). For $i \in I$ and $u \in V(H)$, let $u_i := \alpha(u)$, where $\alpha: V(H) \rightarrow V(H_i)$ is a canonical isomorphism. Let $u, v \in V(G^* \uparrow G)$ be two vertices such that $u \in V(H)$ and uv is a red edge in $G^* \uparrow G$. By Definition 25, such vertices u and v exist, and by definition of vertex integrity, v is a descendant of S . Let $d := 2^{p+1} + 1$. We shall prove by induction that the following claim holds.

▷ **Claim 27.** For each $a \in [0, p-1]$, there is a set $I_a \subseteq I$ of size at least $f(p)/d^{pa+1}$ such that for each $i, j \in I_a$ and each vertex $w \in V(H)$ at distance at most a from u in H , there is a vertex $x \in V(G^*)$ such that $w_i, w_j \in \beta(x)$.

Observe that this statement implies that H_i and H_j for any $i, j \in I_{p-1}$ are merged in G^* because the diameter of H is at most $p-1$.

Proof of Claim 27. Let us start by proving Claim 27 for $a = 0$. Let $U = \{u_i \mid i \in I\}$ and observe that for each $i \in I$, the descendant w'_i of u_i is a red neighbor of v in G^* , by Definition 19 (unless $w'_i = v$). However, the red degree of v in G^* is at most $\text{tw}(G') \leq 2^{p+1}$ (because the treewidth of G' is at most the vertex integrity of G' , and the twin-width is bounded by treewidth, see [32]). Hence, the vertices of U are present in the bags of at most d vertices in G^* (note that some vertices of U may be in the bag of v), which means that there is a vertex $w \in V(G^*)$ with at least $f(p)/d$ vertices of U in its bag. Now it suffices to set $I_0 := \{i \in I \mid u_i \in \beta(w)\}$. This concludes the proof of the base case of the induction.

For the induction step, suppose that Claim 27 holds for some $a \in [0, p-2]$, i.e., there is a set $I_a \subseteq I$ with the described properties. Let $D_a, D_{a+1} \subseteq V(H)$ be the sets of vertices at distance exactly a or $a+1$ from u in H , respectively. Let $w \in D_{a+1}$ and $x \in D_a$ be two neighbors in H . Let x' be the descendant of x_i in G^* for some $i \in I_a$ (or, equivalently, for each $i \in I_a$, by the induction hypothesis), let $W = \{w_i \mid i \in I_a\}$, and let w'_i be the descendant of w_i in G^* (for any $i \in I_a$). Observe that $x'w'_i$ is a red edge of G^+ , unless $x' = w'_i$. Using the same argument as in the base case, x' has red degree at most $d-1$ in G^* , which means that the vertices of W are present in the bags of at most d vertices in G^* .

Since $|D_{a+1}| \leq p$, I_a can be partitioned into at most d^p parts such that if $i, j \in I_a$ are in the same part, then for each vertex $w \in D_{a+1}$, w_i and w_j are in the bag of the same vertex in G^* . Hence, one of these parts has size at least $|I_a|/d^p$, and we choose it to be I_{a+1} . A simple computation shows that I_{a+1} satisfies Claim 27. ◁

Finally, we only need to verify that $|I_{p-1}| \geq f(p)/d^{p(p-1)+1} \geq 2$. Recall that $f(p) = 2^{7p^3}$ and $d = 2^{p+1} + 1 \leq 2^{3p}$ since $p \geq 1$. Since $p(p-1) + 1 \leq 2p^2$, we get $|I_{p-1}| \geq 2^{7p^3}/2^{6p^3} \geq 2$, which concludes the proof. \blacktriangleleft

Now we need to take a closer look at S .

► **Definition 28.** Let $H \in \mathcal{C}$ and $u, v \in S$. We say that u and v are H -equivalent, denoted $u \sim_H v$, if and only if for each $w \in V(H)$, $uw \in E(G) \Leftrightarrow vw \in E(G)$. Let $S^H \subseteq S$ be the set of vertices with at least one neighbor in H (in G). If G'_i is a trigraph in a contraction sequence of a \mathcal{C} -respecting graph, then we denote by S_i^H the set of descendants of S^H in G'_i .

A crucial observation is that before the C' -critical trigraph for H , only very restricted contractions may involve vertices of S^H (so that a red edge to H does not appear).

► **Observation 29.** If G' is a \mathcal{C} -respecting graph, $C' = (G'_1, G'_2, \dots)$ is a contraction sequence of G' , $H \in \mathcal{C}$ is a graph such that $H \not\subseteq G'$, G'_i is the C' -critical trigraph for H , and $j < i$, then for each $u \in S_j^H$, the bag $\beta(u)$ is a subset of an equivalence class of \sim_H .

Proof. Suppose for contradiction that there is $u \in S_j^H$ such that $\beta(u)$ is not a subset of an equivalence class of \sim_H . If $\beta(u) \not\subseteq S$, then clearly all neighbors of u in H (in $G'_j \uparrow G$) would be red, a contradiction with $j < i$ and the choice of i . Hence, assume $\beta(u) \subseteq S$. If there are $v_0, v_1 \in \beta(u)$ such that $v_0 \not\sim_H v_1$, then there is a vertex $w \in H$ that has exactly one neighbor in $\{v_0, v_1\}$ in G , by Definition 28. Thus, uw is a red edge in $G'_j \uparrow G$, again a contradiction \blacktriangleleft

Using Observation 29, we can prove the existence of a C' -safe trigraph.

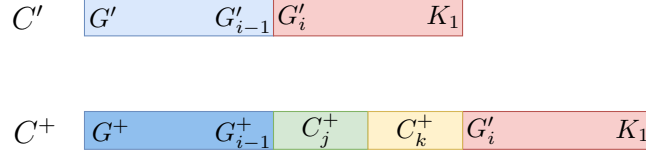
► **Lemma 30.** If G' , C' , H and G'_i are as in Observation 29 and the equivalence class $[H]_{\sim}$ is large in G' , then G'_{i-1} is a C' -safe trigraph for H .

Proof. By Definition 25, it suffices to show that there are two graphs $H', H'' \in [H]_{\sim}$ that are merged in G'_{i-1} . By Lemma 26, we know that such merged graphs H' and H'' exist for G'_i . Let $u, v \in V(G'_{i-1})$ be the two vertices that are contracted in G'_i , and suppose for contradiction that H' and H'' are not merged already in G'_{i-1} . This implies that u and v are both descendants of $H' \cup H''$. However, by Observation 29, $u, v \notin S_{i-1}^H$. This is a contradiction with Definition 25 because the contraction creating G'_i must involve a vertex of S^H so that a red edge incident to H can appear in $G'_i \uparrow G$. \blacktriangleleft

Now we are ready to show how a contraction sequence C' of G' can be modified when a graph $H \in \mathcal{C}$ is added to G' . Unfortunately, we cannot do that without increasing the width. Since our goal is to eventually add many graphs $H \in \mathcal{C}$, we need to keep the increase under control, for which we use the following definition.

► **Definition 31.** A contraction sequence $C = (G_1, \dots, G_n)$ has progressive width $(a \rightarrow_i b)$ if the width of (G_1, \dots, G_{i-1}) is at most a and the width of (G_i, \dots, G_n) is at most b .

► **Lemma 32.** Let G' be a \mathcal{C} -respecting graph, let $C' = (G'_1, G'_2, \dots)$ be a contraction sequence of G' , let $H \in \mathcal{C}$, $H \not\subseteq G'$ be such that $\mathcal{H} := [H]_{\sim}$ is large in G' , let $G^+ = G[V(G') \cup V(H)]$, and let G'_{i-1} be a C' -safe trigraph for H . If C' has progressive width $(t \rightarrow_i 2t)$, then we can construct in polynomial time a contraction sequence C^+ for G^+ of progressive width $(t \rightarrow_i 2t)$. Moreover, if $j < i$, then $G'_j \uparrow G^+$ is the j -th trigraph in C^+ .



■ **Figure 4** A schematic depiction of the construction of C^+ from C' in the proof of Lemma 32. Informally, we insert a new contraction segment after G'_{i-1} (the green and the yellow block), which handles H . The blue prefixes of the two contraction sequences are “morally” the same but in C^+ , H is still present, and so G'_ℓ is not isomorphic to G^+_ℓ for $\ell \in [i-1]$ but it is an induced subtrigraph thereof. On the other hand, the red suffixes are exactly the same since H has been contracted with H' .

Proof. By Definition 25, there are $H', H'' \in \mathcal{H}$ (such that $H', H'' \subseteq G'$) that are merged in G'_{i-1} . Let $\iota: H \rightarrow H'$ be a canonical isomorphism, let $C'_{<i}$ be the prefix of C' of length $i-1$, and let C_H be the partial contraction sequence of H isomorphic to $C'_{<i}[H']$ with an isomorphism induced by ι^2 . Let us now construct $C^+ = (G^+_1 = G^+, G^+_2, \dots)$; see also Figure 4 for an illustration:

1. $C^+_i := (G^+_1, \dots, G^+_{i-1})$ is the extension of $C'_{<i}$ to G^+ , i.e., the same contractions are performed, ignoring H . Note that this construction shows that $G'_j \uparrow G^+$ for each $j < i$, as required.
2. $C^+_j := (G^+_{i-1}, \dots, G^+_j)$ is the extension of C_H to G^+_{i-1} , i.e., C_H is applied to H , ignoring the rest of G^+_{i-1} .
3. Let H_j and H'_j be the subtrigraphs of G^+_j induced by the descendants of H and H' , respectively. By Definition of C_H , there is a bijection α_j from $V(H_j)$ to $V(H'_j)$ that respects³ ι . Let $C^+_k := (G^+_j, \dots, G^+_k)$ be the contraction sequence that contracts u and $\alpha_j(u)$ for every $u \in V(H_j)$ in arbitrary order.
4. We will prove that $G^+_k \cong G'_{i-1}$, and we will define the rest of C^+ to be the suffix of C' starting with G'_i .

Let us argue that C^+ can be computed in polynomial time. First, we need to find the two merged graphs $H', H'' \in \mathcal{H}$: this can be done by brute force because the size of \mathcal{H} is at most $\mathcal{O}(n)$ and checking whether given H' and H'' are merged can be done efficiently (the details depend on the computational model and the representation of contraction sequences). Then, we compute C_H by going through $C'_{<i}$ and looking only at contractions involving vertices of H' . Using C_H , it is easy to compute C^+_j . All other parts of C^+ can clearly be computed in polynomial time.

Now we need to show that C^+ has progressive width ($t \rightarrow_i 2t$). By the assumption about the progressive width of C' , C^+_i has width at most t (using also the fact that G'_{i-1} is a C' -safe trigraph for H ; no red edge in C^+_i is incident to H). Hence, we only need to prove that the suffix of C^+ starting with G^+_i has width at most $2t$. Let S^H_j be the set containing the descendants S^H in G^+_j (or, equivalently, in G'_{i-1} , G^+_{i-1} or G^+_k).

² Formally, an isomorphism from $(G = G_1, \dots, G_n)$ to $(H = H_1, \dots, H_n)$ induced by an isomorphism $\alpha: G \rightarrow H$ is a sequence of isomorphisms $\alpha_i: G_i \rightarrow H_i$ such that for each $i \in [n]$ and $u \in V(G_i)$, $\beta(u) = \alpha^{-1}(\beta(\alpha_i(u)))$.

³ By respecting ι , we mean that if $u \in \beta(v)$ for $u \in V(H)$, $v \in V(H_j)$, then $\iota(u) \in \beta(\alpha_j(v))$.

▷ Claim 33. C_j^+ has width at most $2t$. Moreover, descendants of H have red degree at most t in trigraphs of C_j^+ .

Proof of the Claim. Let $\ell \in [i, j]$, let H_ℓ be the subtrigraph of G_ℓ^+ induced by the descendants of H , and let $m \in [i-1]$ be an index such that the subtrigraph H'_m of G_m^+ induced by the descendants of H' satisfies $|V(H_\ell)| = |V(H'_m)|$. We need to show that the red degree of each $u \in V(G_\ell^+)$ is at most $2t$ (and at most t when $u \in V(H_\ell)$). By construction of C_j^+ , there is a bijection $\alpha : V(H_\ell) \rightarrow V(H'_m)$ such that if $u \in \beta(v)$ for $u \in V(H)$, $v \in V(H_\ell)$, then $\iota(u) \in \beta(\alpha(v))$.

Let $u \in V(H_\ell)$. We will construct a (partial) injection $\gamma : V(G_\ell^+) \rightarrow V(G_m^+)$ such that if $uv \in R(G_\ell^+)$, then $\alpha(u)\gamma(v) \in R(G_m^+)$. Since $\alpha(u)$ has red degree at most t in G_m^+ , this will prove that u has red degree at most t in G_ℓ^+ . Let $v \in V(G_\ell^+)$ be a red neighbor of u in G_ℓ^+ . There are two cases to be considered:

1. If $v \in V(H_\ell)$, then $\alpha(u)\alpha(v) \in R(H'_m)$, using the fact that $\beta(u), \beta(v) \subseteq V(H)$, and we set $\gamma(v) := \alpha(v)$.
2. If $v \notin V(H_\ell)$, then $v \in S_j^H$ by construction of C_j^+ . Let $v_0 \in \beta(v)$. By Observation 29, $\beta(v)$ is a subset of an equivalence class of \sim_H . Hence, there are $u_0, u_1 \in \beta(u)$ such that $u_0v_0 \in E(G)$ but $u_1v_0 \notin E(G)$, and we let $\gamma(v) \in V(G_m^+)$ be the unique vertex such that $v_0 \in \beta(\gamma(v)) \subseteq \beta(v)$.

Now we only need to show that a vertex $v \in S_j^H$ has red degree at most $2t$ in G_ℓ^+ (no other vertex is affected by contractions among descendants of H). Let $K \subseteq V(H_\ell)$ be the set of red neighbors of v in H_ℓ (in G_ℓ^+). By Observation 29, some (actually, each) ancestor $v_0 \in V(G_m^+)$ of v has among its red neighbors all vertices of $\alpha(K)$ in G_m^+ . Since the red degree of v_0 is at most t in G_m^+ and α is a bijection, we obtain that $|K| \leq t$. Hence, v has at most t red neighbors in H_ℓ (in G_ℓ^+). All other red neighbors of v in G_ℓ^+ are its red neighbors also in G_{i-1}^+ (which has maximum red degree at most t), and so v has indeed red degree at most $2t$ in G_ℓ^+ . ◁

▷ Claim 34. C_k^+ has width at most $2t$.

Proof of the Claim. Let $\ell \in [j, k]$, let H_ℓ, H'_ℓ be subtrigraphs of G_ℓ^+ induced by the descendants of H and H' , respectively, let $H_\ell^+ := H_\ell \cup H'_\ell$, and let $\alpha_j : V(H_j) \rightarrow V(H'_j)$ be the bijection defined in the construction of C_k^+ . We need to show that the maximum red degree in G_ℓ^+ is at most $2t$.

First, let $v \in V(G_\ell^+ - H_\ell^+)$. By construction of C_k^+ , we know that $v \in V(G_j^+)$. Suppose that v has higher red degree in G_ℓ^+ than in G_j^+ . This can happen only if a black edge $uv \in E(G_j^+)$ becomes red because of a contraction involving u . However, the only contractions happening in C_k^+ are between u and $\alpha_j(u)$ for some $u \in V(H_j)$, and $uv \in E(G_j^+)$ if and only if $\alpha_j(u)v \in E(G_j^+)$, by definition of α_j . Hence, the red degree of v in G_ℓ^+ is at most its red degree in G_j^+ , and that is at most $2t$ by Claim 33.

Second, we need to show that each $u \in V(H_\ell^+)$ has red degree at most $2t$ in G_ℓ^+ . Observe that H'_ℓ contains no black edges because each vertex $u \in V(H'_\ell)$ is a descendant of both H' and H'' . Hence, a vertex $u \in V(H'_\ell) \setminus V(H_\ell)$ has red degree at most t in G_ℓ^+ because it cannot have higher red degree in G_ℓ^+ than in G_j^+ . Conversely, let $u \in V(H_\ell)$ and let d be the degree of the ancestor $u_0 \in V(H_j)$ of u in H_j . Observe that u has degree at most $2d$ in H_ℓ^+ : for each neighbor $v_0 \in V(H_j)$ of u_0 in H_j , u can have two neighbors in H_ℓ^+ , namely v_0 and $\alpha_j(v_0)$; this happens when u_0 has been contracted with $\alpha_j(u_0)$ into u but no neighbor $v_0 \in V(H_j)$ of u_0 has been contracted with $\alpha_j(v_0)$. Moreover, u_0 and $\alpha_j(u_0)$ have exactly the same red neighbors in S_j^H (by definition of α_j). Hence, the red degree of u in G_ℓ^+ has increased by at most $d \leq t$, compared to the red degree of u_0 in G_j^+ , and so u has at most $t + d \leq 2t$ red neighbors, which concludes the proof. ◁

Since H'_j contains no black edges (each of its vertices is a descendant of both H' and H''), the contraction of H_j and H'_j creates no new red edge (using also the fact that H_j and H'_j are attached to S_j^H in the same way). Hence, we obtain that $G_k^+ \cong G_j^+ - H_j \cong G'_{i-1}$, and we can indeed define the rest of C^+ to be the suffix of C' starting with G'_i . This suffix has width at most $2t$, since C' has progressive width ($t \rightarrow_i 2t$). \blacktriangleleft

Now we are finally ready to prove Lemma 22. This is the only remaining part of this section because we have already shown how Lemma 22 implies Theorem 23, see Subsection 5.1.

Proof of Lemma 22. The idea of the proof is to iteratively apply Lemma 32 to all the graphs in \mathcal{C} not present in the reduced graph G' . However, this requires some care, as applying the lemma in the wrong order might fail to ensure the precondition on the progressive-width. In order to prove this lemma, we will consider the following key claim:

\triangleright **Claim 35.** Given G^* , C^* , and \mathcal{L}^* satisfying the following properties, we can construct in polynomial time a contraction sequence C of width at most $2t$ for G .

1. G^* is a \mathcal{C} -respecting graph;
2. \mathcal{L}^* is a list of pairs (graph H , integer δ), such that the integer value is non-increasing;
3. Each pair (H, δ) in \mathcal{L}^* satisfies all of the following: (i.) $H \in \mathcal{C}$, and H appears only once in \mathcal{L}^* , (ii.) $H \not\subseteq G^*$, (iii.) $[H]_{\sim}$ is large in G^* , (iv.) G_{δ}^* is C^* -safe for H ;
4. C^* is a contraction sequence for G^* of width at most $2t$, and if (H_0, δ_0) is the first pair in \mathcal{L}^* , then C^* has progressive width ($t \rightarrow_{\delta_0+1} 2t$);
5. $V(G^*) \cup \bigcup_{(H, \delta) \in \mathcal{L}^*} V(H) = V(G)$.

Proof of Claim 35. We proceed by induction on the length of \mathcal{L}^* . The base case is trivial: if \mathcal{L}^* is the empty list, the conditions 1. and 5. ensure that $G^* = G$, and 4. ensures that C^* has width $2t$.

Now let us suppose that the claim is true for any list of length i , for some $i \geq 0$. Consider G^* , C^* , \mathcal{L}^* satisfying the hypothesis such that \mathcal{L}^* contains $i + 1$ elements, the first of which being (H_0, δ_0) . We can apply Lemma 32 to G^* , C^* and H_0 since the points 1., 3. and 4. are exactly the preconditions of the lemma, and we obtain in polynomial time a contraction sequence C^+ of progressive width ($t \rightarrow_{\delta_0+1} 2t$) for $G^+ = G[V(G^*) \cup V(H_0)]$.

Now let us consider \mathcal{L}^+ the suffix of \mathcal{L}^* of length i – i.e., we only remove (H_0, δ_0) – and prove that G^+ , C^+ , and \mathcal{L}^+ satisfy the requirements to apply the induction hypothesis.

The first obvious point is that the length of \mathcal{L}^+ is i . Since G^* is \mathcal{C} -respecting and $H \in \mathcal{C}$, we obtain that G^+ is \mathcal{C} -respecting, i.e., it satisfies 1. We can easily verify 5.:

$$V(G^+) \cup \bigcup_{(H, \delta) \in \mathcal{L}^+} V(H) = V(G^*) \cup V(H_0) \cup \bigcup_{(H, \delta) \in \mathcal{L}^+} V(H) = V(G^*) \cup \bigcup_{(H, \delta) \in \mathcal{L}^*} V(H) = V(G).$$

As a suffix of \mathcal{L}^* , \mathcal{L}^+ satisfies 2., and the first three requirements in 3. are also trivially satisfied. To prove the 3.iv., it is necessary to observe two things. First, observe that for each pair $(H, \delta) \in \mathcal{L}^+$, it holds that $G_{\delta}^+ = G_{\delta}^* \uparrow G^+$ since $\delta \leq \delta_0$, by Lemma 32 (the “moreover” part). Second, observe that there is no red edge in G_{δ}^+ that is not already present in G_{δ}^* : indeed, any such red edge would be incident to H_0 by construction of G_{δ}^+ , and its existence would contradict the definition of δ_0 , i.e., the C^* -safeness for H_0 of $G_{\delta_0}^*$. Hence we conclude that for every $(H, \delta) \in \mathcal{L}^+$, it holds that G_{δ}^+ is C^+ -safe for H .

The last item to check, requirement 4., is easily handled: we know that C^+ has progressive width ($t \rightarrow_{\delta_0+1} 2t$) by Lemma 32, and for all $(H, \delta) \in \mathcal{L}^+$, it holds that $\delta \leq \delta_0$ by 2., i.e., by the monotony of \mathcal{L}^* in the second component.

Using the induction hypothesis, we can now create in polynomial time a contraction sequence C of width at most $2t$ for G . The total running time is polynomial, hence the claim is proven. \triangleleft

To finish the proof of Lemma 22, we only need to construct the initial list \mathcal{L}' for G' . For each graph $H \in \mathcal{C}$ such that $H \not\subseteq G'$, let $\delta(H)$ be the index of the last C' -safe trigraph for H , whose existence is ensured by Lemma 30. Let \mathcal{L}' be the list of pairs $(H, \delta(H))$, ordered by non-increasing values of $\delta(H)$, and recall that C' is given. It is easy to see that the requirements on G' , C' and \mathcal{L}' are satisfied – either by definition or by construction – to apply Claim 35: we obtain in polynomial time a contraction sequence C of width at most $2t$ for G , and since the creation of \mathcal{L}' can be achieved in polynomial time, we have proven the lemma. \blacktriangleleft

6 Concluding Remarks

While we believe that the results presented here provide an important contribution to the state of the art in the area of computing twin-width, many prominent questions still remain unanswered. Apart from the “grand prize” – resolving the parameterized approximability of twin-width when the runtime parameter is twin-width itself – future research may focus on finding fixed-parameter algorithms that compute optimal or near-optimal contraction sequences under less restrictive runtime parameters than those considered in this article.

More specifically, the problem remains entirely open when parameterized by treewidth and treedepth, and resolving this may require new insights into the structural properties of optimal contraction sequences and lead to tighter bounds on the twin-width of well-structured graphs. For treedepth in particular, we suspect that combining the ideas presented in Section 5 with the *iterative pruning* approach typically used for treedepth-based algorithms [26, 28, 8] may be an enticing direction to pursue; however, we note that such a combination does not seem straightforward.

References

- 1 Jungho Ahn, Kevin Hendrey, Donggyu Kim, and Sang-il Oum. Bounds for the twin-width of graphs. *SIAM Journal on Discrete Mathematics*, 36(3):2352–2366, 2022. doi:10.1137/21M1452834.
- 2 Jakub Balabán, Robert Ganian, and Mathis Rocton. Computing twin-width parameterized by the feedback edge number. In Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshantov, editors, *41st International Symposium on Theoretical Aspects of Computer Science, STACS 2024, March 12-14, 2024, Clermont-Ferrand, France*, volume 289 of *LIPICs*, pages 7:1–7:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICs.STACS.2024.7.
- 3 Jakub Balabán and Petr Hlinený. Twin-width is linear in the poset width. In Petr A. Golovach and Meirav Zehavi, editors, *16th International Symposium on Parameterized and Exact Computation, IPEC 2021, September 8-10, 2021, Lisbon, Portugal*, volume 214 of *LIPICs*, pages 6:1–6:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.IPEC.2021.6.
- 4 Jakub Balabán, Petr Hlinený, and Jan Jedelský. Twin-width and transductions of proper k -mixed-thin graphs. In Michael A. Bekos and Michael Kaufmann, editors, *Graph-Theoretic Concepts in Computer Science - 48th International Workshop, WG 2022, Tübingen, Germany, June 22-24, 2022, Revised Selected Papers*, volume 13453 of *Lecture Notes in Computer Science*, pages 43–55. Springer, 2022. doi:10.1007/978-3-031-15914-5_4.

- 5 Michael J. Bannister, Sergio Cabello, and David Eppstein. Parameterized complexity of 1-planarity. *J. Graph Algorithms Appl.*, 22(1):23–49, 2018. doi:10.7155/jgaa.00457.
- 6 Pierre Bergé, Édouard Bonnet, and Hugues Déprés. Deciding twin-width at most 4 is np-complete. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPICs*, pages 18:1–18:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ICALP.2022.18.
- 7 Benjamin Bergougnoux, Eduard Eiben, Robert Ganian, Sebastian Ordyniak, and M. S. Ramanujan. Towards a polynomial kernel for directed feedback vertex set. *Algorithmica*, 83(5):1201–1221, 2021. doi:10.1007/s00453-020-00777-5.
- 8 Sujoy Bhore, Robert Ganian, Fabrizio Montecchiani, and Martin Nöllenburg. Parameterized algorithms for queue layouts. *J. Graph Algorithms Appl.*, 26(3):335–352, 2022. doi:10.7155/JGAA.00597.
- 9 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Preprocessing for treewidth: A combinatorial analysis through kernelization. *SIAM J. Discret. Math.*, 27(4):2108–2142, 2013. doi:10.1137/120903518.
- 10 Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width II: small classes. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1977–1996. SIAM, 2021. doi:10.1137/1.9781611976465.118.
- 11 Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width III: max independent set, min dominating set, and coloring. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPICs*, pages 35:1–35:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ICALP.2021.35.
- 12 Édouard Bonnet, Ugo Giocanti, Patrice Ossona de Mendez, Pierre Simon, Stéphan Thomassé, and Szymon Torunczyk. Twin-width IV: ordered graphs and matrices. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 924–937. ACM, 2022. doi:10.1145/3519935.3520037.
- 13 Édouard Bonnet, Ugo Giocanti, Patrice Ossona de Mendez, and Stéphan Thomassé. Twin-width V: linear minors, modular counting, and matrix multiplication. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023, March 7-9, 2023, Hamburg, Germany*, volume 254 of *LIPICs*, pages 15:1–15:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.STACS.2023.15.
- 14 Édouard Bonnet, Eun Jung Kim, Amadeus Reinald, and Stéphan Thomassé. Twin-width VI: the lens of contraction sequences. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 1036–1056. SIAM, 2022. doi:10.1137/1.9781611977073.45.
- 15 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. *J. ACM*, 69(1):3:1–3:46, 2022. doi:10.1145/3486655.
- 16 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
- 17 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
- 18 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

- 19 Pål Grønås Drange, Markus S. Dregi, and Pim van 't Hof. On the computational complexity of vertex integrity and component order connectivity. *Algorithmica*, 76(4):1181–1202, 2016. doi:10.1007/S00453-016-0127-X.
- 20 Pavel Dvorák, Eduard Eiben, Robert Ganian, Dusan Knop, and Sebastian Ordyniak. The complexity landscape of decompositional parameters for ILP: programs with few global variables and constraints. *Artif. Intell.*, 300:103561, 2021. doi:10.1016/J.ARTINT.2021.103561.
- 21 Eduard Eiben, Robert Ganian, Thekla Hamm, Lars Jaffke, and O-joung Kwon. A unifying framework for characterizing and computing width measures. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPICs*, pages 63:1–63:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ITCS.2022.63.
- 22 David Eppstein. The widths of strict outerconfluent graphs. *CoRR*, abs/2308.03967, 2023. arXiv:2308.03967.
- 23 Johannes Klaus Fichte, Robert Ganian, Markus Hecher, Friedrich Slivovsky, and Sebastian Ordyniak. Structure-aware lower bounds and broadening the horizon of tractability for QBF. In *LICS*, pages 1–14, 2023. doi:10.1109/LICS56636.2023.10175675.
- 24 Robert Ganian, Fabian Klute, and Sebastian Ordyniak. On structural parameterizations of the bounded-degree vertex deletion problem. *Algorithmica*, 83(1):297–336, 2021. doi:10.1007/S00453-020-00758-8.
- 25 Robert Ganian and Viktoriia Korchemna. The complexity of bayesian network learning: Revisiting the superstructure. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 430–442, 2021. URL: <https://proceedings.neurips.cc/paper/2021/hash/040a99f23e8960763e680041c601acab-Abstract.html>.
- 26 Robert Ganian and Sebastian Ordyniak. The complexity landscape of decompositional parameters for ILP. *Artif. Intell.*, 257:61–71, 2018. doi:10.1016/J.ARTINT.2017.12.006.
- 27 Robert Ganian and Sebastian Ordyniak. The power of cut-based parameters for computing edge-disjoint paths. *Algorithmica*, 83(2):726–752, 2021. doi:10.1007/s00453-020-00772-w.
- 28 Robert Ganian, Tomás Peitl, Friedrich Slivovsky, and Stefan Szeider. Fixed-parameter tractability of dependency QBF with structural parameters. In Diego Calvanese, Esra Erdem, and Michael Thielscher, editors, *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, KR 2020, Rhodes, Greece, September 12-18, 2020*, pages 392–402, 2020. doi:10.24963/KR.2020/40.
- 29 Tatsuya Gima, Tesshu Hanaka, Masashi Kiyomi, Yasuaki Kobayashi, and Yota Otachi. Exploring the gap between treedepth and vertex cover through vertex integrity. *Theor. Comput. Sci.*, 918:60–76, 2022. doi:10.1016/J.TCS.2022.03.021.
- 30 Tatsuya Gima and Yota Otachi. Extended MSO model checking via small vertex integrity. *Algorithmica*, 86(1):147–170, 2024. doi:10.1007/S00453-023-01161-9.
- 31 Tesshu Hanaka, Michael Lampis, Manolis Vasilakis, and Kanae Yoshiwatari. Parameterized vertex integrity revisited. *CoRR*, abs/2402.09971, 2024. doi:10.48550/arXiv.2402.09971.
- 32 Hugo Jacob and Marcin Pilipczuk. Bounding twin-width for bounded-treewidth graphs, planar graphs, and bipartite graphs. In Michael A. Bekos and Michael Kaufmann, editors, *Graph-Theoretic Concepts in Computer Science - 48th International Workshop, WG 2022, Tübingen, Germany, June 22-24, 2022, Revised Selected Papers*, volume 13453 of *Lecture Notes in Computer Science*, pages 287–299. Springer, 2022. doi:10.1007/978-3-031-15914-5_21.
- 33 Yasuaki Kobayashi and Hisao Tamaki. Treedepth parameterized by vertex cover number. In Jiong Guo and Danny Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*, volume 63 of *LIPICs*, pages 18:1–18:11. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.IPEC.2016.18.

- 34 Michael Lampis and Valia Mitsou. Fine-grained meta-theorems for vertex integrity. In Hee-Kap Ahn and Kunihiko Sadakane, editors, *32nd International Symposium on Algorithms and Computation, ISAAC 2021, December 6-8, 2021, Fukuoka, Japan*, volume 212 of *LIPICs*, pages 34:1–34:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ISAAC.2021.34.
- 35 Sang-il Oum. Rank-width and vertex-minors. *J. Comb. Theory, Ser. B*, 95(1):79–100, 2005. doi:10.1016/j.jctb.2005.03.003.
- 36 Neil Robertson and Paul D. Seymour. Graph minors. i. excluding a forest. *J. Comb. Theory B*, 35(1):39–61, 1983. doi:10.1016/0095-8956(83)90079-5.
- 37 Johannes Uhlmann and Mathias Weller. Two-layer planarization parameterized by feedback edge set. *Theor. Comput. Sci.*, 494:99–111, 2013. doi:10.1016/j.tcs.2013.01.029.
- 38 Martijn van Ee. Some notes on bounded starwidth graphs. *Inf. Process. Lett.*, 125:9–14, 2017. doi:10.1016/J.IPL.2017.04.011.


On the Parameterized Complexity of Eulerian Strong Component Arc Deletion

Václav Blažej  

University of Warwick, Coventry, UK

Satyabrata Jana  

University of Warwick, Coventry, UK

M. S. Ramanujan  

University of Warwick, Coventry, UK

Peter Strulo  

University of Warwick, Coventry, UK

Abstract

In this paper, we study the Eulerian Strong Component Arc Deletion problem, where the input is a directed multigraph and the goal is to delete the minimum number of arcs to ensure every strongly connected component of the resulting digraph is Eulerian.

This problem is a natural extension of the Directed Feedback Arc Set problem and is also known to be motivated by certain scenarios arising in the study of housing markets. The complexity of the problem, when parameterized by solution size (i.e., size of the deletion set), has remained unresolved and has been highlighted in several papers. In this work, we answer this question by ruling out (subject to the usual complexity assumptions) a fixed-parameter tractable (FPT) algorithm for this parameter and conduct a broad analysis of the problem with respect to other natural parameterizations. We prove both positive and negative results. Among these, we demonstrate that the problem is also hard (W[1]-hard or even para-NP-hard) when parameterized by either treewidth or maximum degree alone. Complementing our lower bounds, we establish that the problem is in XP when parameterized by treewidth and FPT when parameterized either by both treewidth and maximum degree or by both treewidth and solution size. We show that these algorithms have near-optimal asymptotic dependence on the treewidth assuming the Exponential Time Hypothesis.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Parameterized complexity, Eulerian graphs, Treewidth

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.4

Related Version *Full Version*: <https://arxiv.org/abs/2408.13819> [2]

Funding Supported by the Engineering and Physical Sciences Research Council (grant number EP/V044621/1).

1 Introduction

In the Eulerian Strong Component Arc Deletion (ESCAD) problem, where the input is a directed graph (digraph)¹ and a number k and the goal is to delete at most k arcs to ensure every strongly connected component of the resulting digraph is Eulerian. This problem was

¹ In this paper, the arc set of a digraph is a multiset, i.e., we allow multiarcs. Moreover, we treat multiarcs between the same ordered pairs of vertices as distinct arcs in the input representation of all digraphs. Consequently, the number of arcs in the input is upper bounded by the length of the input. We exclude loops as they play no non-trivial role in instances of this problem.



first introduced by Cechlárová and Schlotter [3] to model problems arising in the study of housing markets and they left the existence of an FPT algorithm for ESCAD as an open question.

The ESCAD problem extends the well-studied Directed Feedback Arc Set (DFAS) problem. In DFAS, the goal is to delete the minimum number of arcs to make the digraph acyclic. The natural extension of DFAS to ESCAD introduces additional complexity as we aim not to prevent cycles, but aim to balance in-degrees and out-degrees within each strongly connected component. As a result, the balance requirement complicates the problem significantly and the ensuing algorithmic challenges have been noted in multiple papers [3, 6, 11].

Crowston et al. [4] made partial progress on the problem by showing that ESCAD is fixed-parameter tractable (FPT) on tournaments and also gave a polynomial kernelization. However, the broader question of fixed-parameter tractability of ESCAD on general digraphs has remained unresolved.

Our contributions. Our first main result rules out the existence of an FPT algorithm for ESCAD under the solution-size parameterization, subject to standard complexity-theoretic assumptions.

► **Theorem 1.** *ESCAD is $W[1]$ -hard parameterized by the solution size.*

The above negative result explains, in some sense, the algorithmic challenges encountered in previous attempts at showing tractability and shifts the focus toward alternative parameterizations. However, even here, we show that a strong parameterization such as the vertex cover number is unlikely to lead to a tractable outcome.

► **Theorem 2.** *ESCAD is $W[1]$ -hard parameterized by the vertex cover number of the graph.*

In fact, assuming the Exponential Time Hypothesis (ETH), we are able to obtain a stronger lower bound.

► **Theorem 3.** *There is no algorithm solving ESCAD in $f(k) \cdot n^{o(k/\log k)}$ time for some function f , where k is the vertex cover number of the graph and n is the input length, unless the Exponential Time Hypothesis fails.*

To add to the hardness results above, we also analyze the parameterized complexity of the problem parameterized by the maximum degree of the input digraph and show that even for constant values of the parameter, the problem remains NP-hard.

► **Theorem 4.** *ESCAD is NP-hard in digraphs where each vertex has (in, out) degrees in $\{(1, 6), (6, 1)\}$.*

We complement these negative results by showing that ESCAD is FPT parameterized by the treewidth of the deoriented digraph (i.e., the underlying undirected multigraph) and solution size as well as by the treewidth and maximum degree of the input digraph. Furthermore, we give an XP algorithm parameterized by treewidth alone. All three results are obtained by a careful analysis of the same algorithm stated below.

► **Theorem 5.** *An ESCAD instance $\mathcal{I} = (G, k)$ can be solved in time $2^{\mathcal{O}(\text{tw}^2)} \cdot (2\alpha + 1)^{2\text{tw}} \cdot n^{\mathcal{O}(1)}$ where tw is the treewidth of deoriented G , Δ is the maximum degree of G , and $\alpha = \min(k, \Delta)$.*

In the above statement, notice that α is upper bounded by the number of edges in the digraph and so, implies an XP algorithm parameterized by the treewidth with running time $2^{\mathcal{O}(\text{tw}^2)} \cdot n^{\mathcal{O}(\text{tw})}$. Notice the running time of our algorithm asymptotically almost matches our ETH based lower bound (recall that the vertex cover number of a graph is at least the treewidth) in Theorem 3.

Recall that in general, multiarcs are permitted in an instance of ESCAD. This fact is crucially used in the proof of Theorem 2 and raises the question of adapting this reduction to *simple digraphs* (digraphs without multiarcs or loops) in order to obtain a similar hardness result parameterized by vertex cover number. However, we show that this is not possible by giving an FPT algorithm for the problem on simple digraphs parameterized by the vertex integrity of the input graph. Recall that a digraph has *vertex integrity* k if there exists a set of vertices of size $q \leq k$ which when removed, results in a digraph where each weakly connected component has size at most $k - q$. Vertex integrity is a parameter lower bounding vertex cover number and has gained popularity in recent years as a way to obtain FPT algorithms for problems that are known to be W[1]-hard parameterized by treedepth – one example being ESCAD on simple graphs as we show in this paper (see Theorem 7 below).

► **Theorem 6.** *ESCAD on simple digraphs is FPT parameterized by the vertex integrity of the graph.*

As a consequence of this result, we infer an FPT algorithm for ESCAD on simple digraphs parameterized by the vertex cover number, highlighting the difference in the behaviour of the ESCAD problem on directed graphs that permit multiarcs versus simple digraphs. On the other hand, we show that even on simple digraphs this positive result does not extend much further to well-studied width measures such as treewidth (or even the larger parameter treedepth), by obtaining the following consequence of Theorems 2 and 3.

► **Theorem 7.** *ESCAD even on simple digraphs is W[1]-hard parameterized by k and assuming ETH, there is no algorithm solving it in $f(k)n^{(k/\log k)}$ time for some function f , where k is the size of the smallest vertex set that must be deleted from the input digraph to obtain a disjoint union of directed stars and n is the input length.*

Related Work. The vertex-deletion variant of ESCAD is known to be W[1]-hard, as shown by Göke et al. [11], who identify ESCAD as an open problem and note that gaining more insights into its complexity was a key motivation for their study. Cygan et al. [6] gave the first FPT algorithm for edge (arc) deletion to Eulerian graphs (respectively, digraphs). Here, the aim is to make the whole graph Eulerian whereas the focus in ESCAD is on each strongly connected component. Cygan et al. also explicitly highlight ESCAD as an open problem and a motivation for their work. Goyal et al. [12] later improved the algorithm of Cygan et al. by giving algorithms achieving a single-exponential dependence on k .

2 Preliminaries

For a digraph G , we denote its vertices by $V(G)$, arcs by $E(G)$, the subgraph induced by $S \subseteq V(G)$ as $G[S]$, a subgraph with subset of vertices removed as $G - S = G[V(G) \setminus S]$, and a subgraph with subset of edges $F \subseteq E(G)$ removed as $G - F = (V(G), E(G) \setminus F)$. For a vertex v and digraph G , let $\deg_G^-(v)$ denote its in-degree, $\deg_G^+(v)$ be its out-degree, and $\deg_G^+(v) - \deg_G^-(v)$ is called its *imbalance*. If the imbalance of v is 0 then v is said to be *balanced* (in G). A digraph is called *balanced* if all its vertices are balanced. The maximum degree of a digraph G is the maximum value of $\deg_G^+(v) + \deg_G^-(v)$ taken over every vertex v in the graph.

A vertex v is *reachable* from u if there exists a directed path from u to v in G . A *strongly connected component* of G is a maximal set of vertices where all vertices are mutually reachable. Let *strong subgraph* denoted $\text{strong}(G)$ be the subgraph of G obtained by removing all arcs that have endpoints in different strongly connected components. The ESCAD problem can now be formulated as “Is there a set $S \subseteq V(G)$ of size $|S| \leq k$ such that $\text{strong}(G - S)$ is balanced?” We call an arc $e \in E(G)$ *active* in G if $e \in E(\text{strong}(G))$ and *inactive* in G otherwise.

A graph G has vertex cover k if there exists a set of vertices $S \subseteq V(G)$ with bounded size $|S| \leq k$ such that $G - S$ is an independent set. A star is an undirected graph isomorphic to K_1 or $K_{1,t}$ for some $t \geq 0$ and a *directed star* is just a digraph whose underlying undirected graph is a star.

A *tree decomposition* of an undirected graph G is a pair $(T, \{X_t\}_{t \in V(T)})$ where T is a tree and $X_t \subseteq V(G)$ such that (i) for all edges $uv \in E(G)$ there exists a node $t \in V(T)$ such that $\{u, v\} \subseteq X_t$ and (ii) for all $v \in V(G)$ the subgraph induced by $\{t \in V(T) : v \in X_t\}$ is a non-empty tree. The *width* of a tree decomposition is $\max_{t \in V(T)} |X_t| - 1$. The *treewidth* of G is the minimum width of a tree decomposition of G .

Let $(T, \{X_t\}_{t \in V(T)})$ be a tree decomposition of G . We refer to every node of T with degree one as a *leaf node* except one which is chosen as the root, r . A tree decomposition $(T, \{X_t\}_{t \in V(T)})$ is a *nice tree decomposition with introduce edge nodes* if all of the following conditions are satisfied:

1. $X_r = \emptyset$ and $X_\ell = \emptyset$ for all leaf nodes ℓ .
2. Every non-leaf node of T is one of the following types:
 - **Introduce vertex node:** a node t with exactly one child t' such that $X_t = X_{t'} \cup v$ for some vertex $v \notin X_{t'}$.
 - **Introduce edge node:** a node t , labeled with an edge uv where $u, v \in X_t$ and with exactly one child t' such that $X_t = X_{t'}$.
 - **Forget node:** a node t with exactly one child t' such that $X_t = X_{t'} \setminus \{v\}$ for some vertex $v \in X_{t'}$.
 - **Join node:** a node t with exactly two children t_1, t_2 such that $X_t = X_{t_1} = X_{t_2}$.
3. Every edge appears on exactly one introduce edge node.

Proofs for results marked \star can be found in the full version [2].

3 Our Results for ESCAD

In the following four subsections we describe three hardness results and tractability results on bounded treewidth graphs for ESCAD. In Section 3.4 we show that the problem is XP by treewidth and FPT in two cases – when parameterized by the combined parameter treewidth plus maximum degree, and when parameterized by treewidth plus solution size. The hardness results show that dropping any of these parameters leads to a case that is unlikely to be FPT. More precisely, we show that parameterized by solution size it is W[1]-hard (in Section 3.1) as is the case when parameterized by vertex cover number (Section 3.2), and it is para-NP-hard when parameterized by the maximum degree (Section 3.3).

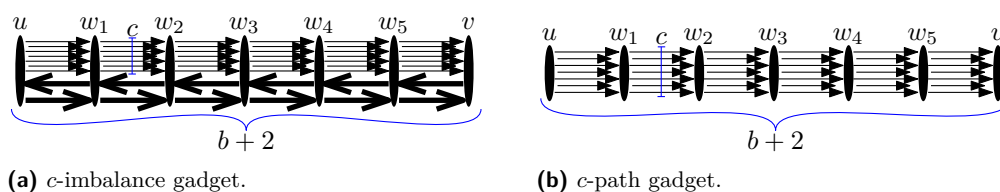
3.1 W[1]-hardness of ESCAD Parameterized by Solution Size

In this section, we show that ESCAD is W[1]-hard when parameterized by solution size. Our reduction is from MULTICOLORED CLIQUE. The input to MULTICOLORED CLIQUE consists of a simple undirected graph G , an integer ℓ , an containing exactly one vertex from each

set $V_i, i \in [\ell]$. MULTICOLORED CLIQUE is known to be $W[1]$ -hard when parameterized by the size of the solution ℓ [5]. Each set V_i for $i \in [\ell]$ is called a color class and for a vertex v in G , we say v has color i if $v \in V_i$. We assume without loss of generality that in the MULTICOLORED CLIQUE instance we reduce from, each color class V_i forms an independent set (edges in the same color class can be removed) and moreover, for each vertex $v \in V_i$ and each $j \in [\ell] \setminus \{i\}$ there exists a $w \in V_j$ that is adjacent to v (any vertex that cannot participate in a multicolored clique can be removed).

We start with descriptions of two auxiliary gadgets: the *imbalance gadget* and the *path gadget*.

Imbalance Gadget. Let u, v be a pair of vertices, and b, c be two positive integers. We construct a gadget $I_{u,v}$ connecting the vertex u to v by a path with vertices u, w_1, \dots, w_b, v where w_i 's are b new vertices (we call them intermediate vertices in this gadget); let $u = w_0$ and $v = w_{b+1}$. For every $i \in \{0, \dots, b\}$ the path contains $b+1+c$ forward arcs (w_i, w_{i+1}) and $b+1$ backward arcs (w_{i+1}, w_i) , see Figure 1a for an illustration. Observe that with respect to the gadget $I_{u,v}$, the vertices u and v have imbalances c and $-c$, respectively, whereas other vertices in the gadget have imbalance zero. We refer to this gadget $I_{u,v}$ as a (b, c) -imbalance gadget.



■ **Figure 1** Black ellipses are vertices, thick edges represent $(b+1)$ copies of the edges; $(b+2)$ is the number of vertices in the gadgets.

Path Gadget. Let u, v be a pair of vertices, and b, c be two positive integers. We construct a gadget $P_{u,v}$ connecting the vertex u to v by a path with vertices u, w_1, \dots, w_b, v where w_i 's are b new intermediate vertices; let $u = w_0$ and $v = w_{b+1}$. For every $i \in \{0, \dots, b\}$ the path contains c forward arcs (w_i, w_{i+1}) . See Figure 1b for an illustration. Notice that, unlike the imbalance gadget, we do not add backward arcs. Observe that with respect to the gadget $P_{u,v}$, the vertices u and v has imbalances c and $-c$, respectively, whereas the other vertices in the gadget have imbalance zero. We refer to this gadget $P_{u,v}$ as a (b, c) -path gadget.

We use the following properties of the gadgets I_{uv} and P_{uv} to reason about the correctness of our construction.

► **Lemma 8** (★). *Let (G, b) be a yes-instance of ESCAD and S be a solution. Assume that for a pair of vertices u, v in G , there is a (b, c) -imbalance gadget I_{uv} present in G (i.e., I_{uv} is an induced subgraph of G). If S is an inclusionwise minimal solution then S contains no arc of I_{uv} .*

► **Lemma 9** (★). *Let (G, b) be a yes-instance of ESCAD and S be an inclusionwise minimal solution for this instance. Assume that for a pair of vertices u, v in G , there is a (b, c) -path gadget P_{uv} present in G (i.e., P_{uv} is an induced subgraph of G) and there are more than b arc-disjoint paths from v to u . If S contains an arc from P_{uv} , then there exists $i \in \{0, \dots, b+1\}$ such that S contains every (w_i, w_{i+1}) arc in P_{uv} .*

Brief idea of the reduction. The main idea of the following reduction is to “choose” vertices and edges of the clique using cuts. First, we enforce an imbalance using (b, c) -imbalance gadgets where b is the budget and let it propagate using path gadgets in a way that chooses a vertex for each color. For each chosen vertex, the solution is then forced to select $(\ell - 1)$ out-going arcs that are incident to it. Choosing the same edge from two sides results in a specific vertex to be cut from the strongly connected component of the remaining graph, decreasing the degree by the correct amount. Our solution creates a set of $\binom{\ell}{2}$ vertices that have out-degree two – these vertices represent edges of the multicolored clique.

► **Theorem 1.** *ESCAD is $W[1]$ -hard parameterized by the solution size.*

Proof. Consider an instance $\mathcal{I} = (G, \ell, (V_1, \dots, V_\ell))$ of MULTICOLORED CLIQUE with n vertices. Recall our assumption that each color class induces an independent set, and every vertex has at least one neighbor in every color class distinct from its own. In polynomial time, we construct an ESCAD instance $\mathcal{I}' = (G', k)$ in the following way (see Figure 2 for an overview).

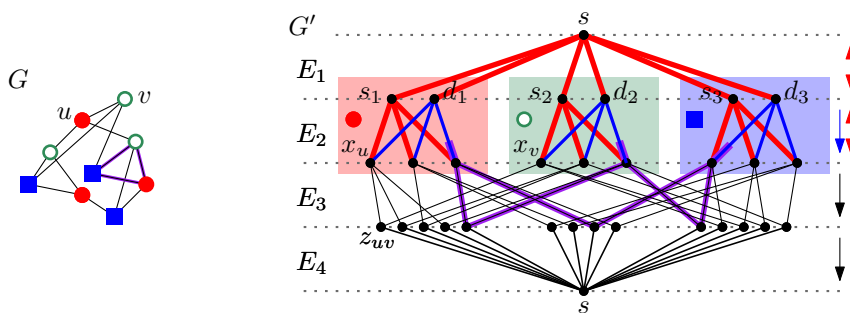
- We set $k = 2\ell(\ell - 1)$.
- Construction of $V(G')$ is as follows:
 1. We add a vertex s .
 2. For each color $j \in [\ell]$, we have a pair of vertices s_j and d_j .
 3. For each vertex u in $V(G)$, we have a vertex x_u .
 4. For each edge uv in $E(G)$, we have a vertex z_{uv} .
- The construction of $E(G')$ is as follows. We introduce four sets of arcs E_1, E_2, E_3 , and E_4 that together comprise the set $E(G')$. For each color $j \in [\ell]$, let $r_j := |V_j| \cdot (\ell - 1)$, $c_j := |\{uv : uv \in E(G), u \in V_j\}| - r_j$. Notice that $c_j \geq 0$ since every vertex in G has degree at least $\ell - 1$.
 1. For each $j \in [\ell]$, we add a $(k, r_j - \ell + 1)$ -imbalance gadget I_{s, d_j} and a (k, c_j) -imbalance gadget I_{s, s_j} to E_1 .
 2. For each $j \in [\ell]$, for each vertex $u \in V_j$ we add a $(k, |N_G(u)| - \ell + 1)$ -imbalance gadget I_{s_j, x_u} and a $(k, \ell - 1)$ -path gadget P_{d_j, x_u} to E_2 .
 3. For every edge $uv \in E(G)$, we add a pair of arcs (x_u, z_{uv}) and (x_v, z_{uv}) to E_3 .
 4. For every edge $uv \in E(G)$, we add two copies of the arc (z_{uv}, s) to E_4 .

It is easy to see that the construction can be performed in time polynomial in $|V(G)|$. Now, we prove the correctness of our reduction. First, we argue about the imbalances of vertices in G' . As each vertex of G' lies on a cycle that goes through s , it follows that G' is strongly connected.

▷ **Claim 10** (\star). The only vertices with non-zero imbalance in G' are those in the set $\{s\} \cup \{d_j : j \in [\ell]\}$. Furthermore, the imbalance of the vertex s is $-\ell(\ell - 1)$ and the imbalance of d_j for each $j \in [\ell]$ is $(\ell - 1)$.

This shows that there are only $\ell + 1$ vertices with non-zero imbalance in G' . The imbalance of the d_j 's will make us “choose” vertices and edges that represent a clique in G as we will see later.

We now show correctness of our reduction. In the forward direction, assume that (G, ℓ) is a yes-instance and let K be a multicolored clique of size ℓ in G . Let v_j denote the vertex with color j in K . We now construct a solution S of (G', k) . For each edge $v_i v_j$ we add the arcs $(x_{v_j}, z_{v_j v_i})$ and $(x_{v_i}, z_{v_j v_i})$ to S . There are $2 \cdot \binom{\ell}{2}$ many such arcs. Now for each $j \in [\ell]$ we add all the incoming arcs of x_{v_j} along the path gadget $P_{d_j x_{v_j}}$ to S . As for each $j \in [\ell]$,



■ **Figure 2** Overview of the reduction from G to G' ; four sets of edges are depicted from top to bottom. E_1 contains imbalance gadgets, E_2 is a mixture of imbalance and path gadgets, E_3 has directed edges, and E_4 has all directed double edges to s . Marked purple edges corresponds to a solution in G and its respective solution in G' . The three colored backgrounds in G' signify part of the construction tied to the three color classes. All edges of the picture of G' are oriented from top to bottom. The picture of G' wraps up as the vertex s drawn on the bottom is the same as the one drawn on the top.

the number of such arcs is $(\ell - 1)$ we have $|S| = 2 \cdot \binom{\ell}{2} + \ell(\ell - 1) = 2\ell(\ell - 1) = k$. Now, we show that each strongly connected component in $G' - S$ is Eulerian. For an example of S , refer to Figure 2 (purple arcs).

We consider the strongly connected components of $G' - S$ and we will show that each of them is Eulerian. We first define:

$$Z = \{z_{uw} : u, w \in K\} \cup \left(\bigcup_{j \in [\ell]} (V(P_{d_j, x_{v_j}}) \setminus \{d_j, x_{v_j}\}) \right)$$

▷ **Claim 11** (★). One strongly connected component of $G' - S$ consists of all the vertices except Z (we call it the *large* component) and all other strongly connected components of $G' - S$ are singleton – one for each vertex in Z .

Since singleton strongly connected components are always balanced, we only need to show that the large component is Eulerian i.e., it is balanced inside the strongly connected component itself.

▷ **Claim 12** (★). The large component is Eulerian

The claim can be proved through a case analysis going through the various types of vertices that appear in this strongly connected component, i.e., the vertex s , the vertices in $\{s_j : j \in [\ell]\} \cup \{z_{uv} : u \notin K \text{ or } v \notin K\} \cup \{x_u : u \notin K\}$, the vertex d_j for $j \in [\ell]$, and x_u where $u \in K$.

This completes the argument in the forward direction.

In the converse direction, assume that (G', k) is a yes-instance and let S be a solution. Let us first establish some structure on S , from which it will be possible to recover a multicolored clique for G .

Let \mathcal{C} denote the strongly connected component of $G' - S$ that contains s . Due to Lemma 8, we may assume that S does not contain any arcs of any of the imbalance gadgets. This implies that \mathcal{C} contains s_j and d_j for every $j \in [\ell]$ as well as x_u for every $u \in V(G)$. Moreover, due to Lemma 9, we know that if S contains arcs of a path gadget P_{d_j, x_u} , then they form a cut in it. As all inclusion-wise minimal cuts of the path gadgets are of the same cardinality and adding any minimal cut of a path gadget to S makes all arcs of the path gadget inactive in $G' - S$, assume that if a cut of a path gadget P_{d_j, x_u} is in S , then the cut consists of the incoming-arcs of x_u in the gadget.

Recall from Claim 10, that the only imbalanced vertices in G' are $\{s\} \cup \{d_j : j \in [\ell]\}$. Let us make some observations based on the fact that these vertices are eventually balanced in $\text{strong}(G' - S)$.

For each $j \in [\ell]$, since none of the incoming arcs of d_j are in S (they lie in an imbalance gadget), in order to make d_j balanced it must be the case that S contains a cut of exactly one of the path gadgets starting at d_j , call it $P_{d_j, x_{v_j}}$. Recall that x_{v_j} was originally balanced in G' . Further, recall that we have argued that x_{v_j} is in \mathcal{C} along with s_j and d_j . Since the imbalance gadget starting at s_j and ending at x_{v_j} cannot intersect S and we have deleted all of the $\ell - 1$ incoming arcs to x from the path gadget $P_{d_j, x_{v_j}}$, the imbalance of $-\ell + 1$ thus created at x_{v_j} needs to be resolved by making exactly $(\ell - 1)$ of its outgoing arcs in E_3 inactive in $G' - S$. Since we have already spent a budget of $\ell(\ell - 1)$ from the path gadgets, the budget that remains to be used for resolving these imbalances at $\{x_{v_j} : j \in [\ell]\}$ is $\ell(\ell - 1)$.

On the other hand, recall that s is imbalanced in G' and to make s balanced, we need to make $\ell(\ell - 1)$ incoming arcs of s (from E_4) inactive in $G' - S$. This is because all outgoing arcs of s lie in imbalance gadgets and cannot be in S .

And finally, recall that for each $uv \in E(G)$, the vertex z_{uv} is balanced in G' (by Claim 10). Since the strongly connected component \mathcal{C} in $G' - S$ contains the vertices s, x_u, x_v (i.e., all neighbors of $z_{u,v}$), for the vertex z_{uv} to remain balanced in $\text{strong}(G' - S)$, we have the following exhaustive cases regarding the arcs between $s, x_u, x_v, z_{u,v}$: (1) none of the four arcs incident to z_{uv} is in S ; (2) one incoming and one outgoing arc are in S ; (3) both incoming arcs or both outgoing arcs are in S . In Case (2), two arcs are added to S , which makes two arcs inactive while in Case (3) two arcs are added to S which makes four arcs inactive. As previously noted, we still need $\ell(\ell - 1)$ arcs in E_3 and $\ell(\ell - 1)$ arcs in E_4 to become inactive in $G' - S$. The required number of inactive arcs in $E_3 \cup E_4$ is twice the remaining budget, so for every z_{uv}, x_u, x_v , the arcs between $s, x_u, x_v, z_{u,v}$ must be in Case (1) or Case (3). Moreover, whenever Case (3) occurs, we may assume without loss of generality that the arcs in S are the two arcs (x_u, z_{uv}) and (x_v, z_{uv}) . Thus, there are exactly $\binom{\ell}{2}$ vertices z_{uv} such that the arcs between $s, x_u, x_v, z_{u,v}$ are in Case (3).

We now extract the solution clique K for (G, ℓ) by taking, for each $j \in [\ell]$, the vertex $v_j \in V(G)$ such that a cut of $P_{d_j, x_{v_j}}$ is contained in S . We have shown that there are exactly $\binom{\ell}{2}$ vertices z_{uv} such that the arcs between $s, x_u, x_v, z_{u,v}$ are in Case (3) and for each $j \in [\ell]$ and the vertex x_{v_j} , exactly $\ell - 1$ of its outgoing arcs are made inactive by S . This can only happen if for every $j, j' \in [\ell]$, there is a vertex $z_{v_j v_{j'}}$, implying that $v_j v_{j'}$ is an edge in G . ◀

3.2 W[1]-hardness of ESCAD Parameterized by Vertex Cover Number

In this section, we show that ESCAD is W[1]-hard when parameterized by the vertex cover number. Jansen, Kratsch, Marx, and Schlotter [13] showed that UNARY BIN PACKING is W[1]-hard when parameterized by the number of bins h .

UNARY BIN PACKING

Input: A set of positive integer item sizes x_1, \dots, x_n encoded in unary, a pair of integers h and b .

Question: Is there a partition of $[n]$ into h sets J_1, \dots, J_h such that $\sum_{\ell \in J_j} x_\ell \leq b$ for every $j \in [h]$?

In order to carefully handle vertex balances in our reduction, it is helpful to work with a variant of the above problem, called EXACT UNARY BIN PACKING, where the inequality $\sum_{\ell \in J_j} x_\ell \leq b$ is replaced with the equality $\sum_{\ell \in J_j} x_\ell = b$. That is, in this variant, all bins get filled up to their capacity.

► **Theorem 2.** *ESCAD is W[1]-hard parameterized by the vertex cover number of the graph.*

Proof. Let $\mathcal{I}' = ((x_1, \dots, x_m), h, b)$ be an instance of UNARY BIN PACKING. If $b \geq \sum_{i=1}^m x_i$, then \mathcal{I}' is trivially a yes-instance and we can return a trivial yes-instance of ESCAD with vertex cover number at most h . In the same way, if $b \cdot h < \sum_{i \in [m]} x_i$, then \mathcal{I}' is trivially a no-instance and we return a trivial no-instance of ESCAD with vertex cover number at most h . Now, suppose neither of the above cases occur.

Note that the length of the unary encoding of b is upper bounded by the total length of the unary encoding of all items x_1, \dots, x_m . Similarly, if $h \geq m$ then the instance boils down to checking whether $x_i \leq b$ for every $i \in [m]$ (and producing a trivial ESCAD instance accordingly) so we can assume that $h < m$, hence, the length of the unary encoding of h is upper bounded by the total length of the unary encoding of all items. We now construct an instance \mathcal{I} of EXACT UNARY BIN PACKING from \mathcal{I}' by adding $h \cdot b - \sum_{i \in [m]} x_i$ one-sized items (this is non-negative because of the preprocessing steps). If \mathcal{I}' is a yes-instance, then one can fill-in the remaining capacity in every bin with the unit-size items, to get a solution for \mathcal{I} . Conversely, if \mathcal{I} is a yes-instance, then removing the newly added unit-size items yields a solution for \mathcal{I}' . Let n denote the number of items in \mathcal{I} . Note that since $\sum_{i \in [n]} x_i = b \cdot h$, this implies that $|\mathcal{I}| = \mathcal{O}(|\mathcal{I}'|^2)$, the instance of EXACT UNARY BIN PACKING remains polynomially bounded.

We next reduce the EXACT UNARY BIN PACKING instance \mathcal{I} to an instance $\mathcal{I}^* = (G, k)$ of ESCAD in polynomial time. Let us fix the budget $k = b \cdot h(h - 1)$. We now build a graph G that models the bins by k copies of interconnected gadgets (that form the vertex cover) and models each item as a vertex of the independent set. In our reduction, we use the following terms. For a pair of vertices p, q , a c -arc (p, q) denotes c parallel copies of the arc (p, q) and a thick arc (p, q) denotes a $3k$ -arc (p, q) . The construction of G is as follows.

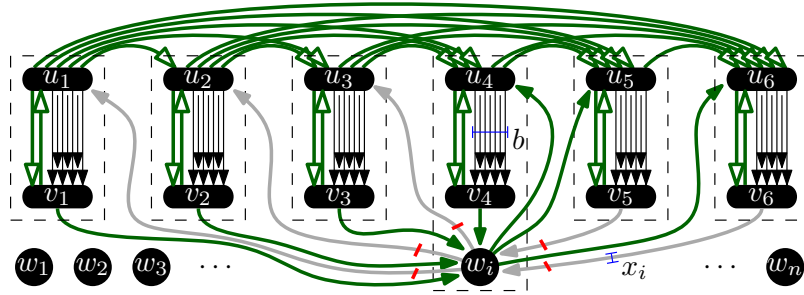
- The vertex set of G is the set $\{u_j : j \in [h]\} \cup \{v_j : j \in [h]\} \cup \{w_i : i \in [n]\}$.
- For each $j \in [h]$, we add a b -arc (u_j, v_j) , a thick arc (u_j, v_j) and a thick arc (v_j, u_j) . We call the subgraph induced by u_j, v_j and these arcs, the b -imbalance gadget B_j .
- Next, we add thick arcs $(u_j, u_{j'})$ for every $j < j'$ where $j, j' \in [h]$.
- Finally, for each $i \in [n]$ and $j \in [h]$, we add x_i -arcs (w_i, u_j) and (v_j, w_i) .

This concludes the construction, see Figure 3. Before we argue the correctness, let us make some observations.

Note that the vertices participating in the imbalance gadgets form a vertex cover of the resulting graph and their number is upper bounded by $2h$. Hence, if we prove the correctness of the reduction, we have the required parameterized reduction from UNARY BIN PACKING parameterized by the number of bins to ESCAD parameterized by the vertex cover number of the graph.

We say that a set of arcs in G cuts a (p, q) arc if it contains all parallel copies of (p, q) . Note that no set of at most k arcs cuts a thick (p, q) arc. In particular, no solution to the ESCAD instance (G, k) cuts any thick arc (p, q) that appears in the graph.

Exact Unary Bin Packing is a yes-instance \Rightarrow ESCAD is a yes-instance. Assume that we have a partition J_1, \dots, J_h that is a solution to \mathcal{I} . We now define a solution S for \mathcal{I}^* . For every $x_i \in J_j$ we cut (i.e., add to S) all parallel copies of the arc $(w_i, u_{j'})$ for every $j' < j$ and



■ **Figure 3** A part of the resulting ESCAD instance after reduction from EXACT UNARY BIN PACKING with six bins; connections between the independent vertices and imbalance gadgets are shown only for one vertex w_i . Thick arcs are shown with empty arrowhead, bold arcs incident to w_i are x_i -arcs. Crossed off arcs are in a solution and dashed boxes show strongly connected components of the solution. This example represents $x_i \in J_4$.

we cut all parallel copies of the arc $(v_{j''}, w_i)$ for every $j'' > j$. This results in cutting a total of $x_i \cdot (h - 1)$ arcs incident to each w_i and as $\sum_{i=1}^n x_i = b \cdot h$ we cut exactly $b \cdot h(h - 1) = k$ arcs in total.

▷ **Claim 13** (★). $\text{strong}(G - S)$ is balanced.

ESCAD is a yes-instance \Rightarrow Exact Unary Bin Packing is a yes-instance. We aim to show that in any solution for the ESCAD instance, the arcs that are cut incident to w_i for any $i \in [n]$ have the same structure as described in the other direction, i.e., for all w_i there exists j such that the solution cuts $(w_i, u_{j'})$ for all $j' < j$ and it cuts $(v_{j''}, w_i)$ for all $j'' > j$. This is equivalently phrased in the following claim.

▷ **Claim 14** (★). There are no two indices $a, b \in [h]$ with $a < b$ such that both (w_i, u_a) and (v_b, w_i) are uncut.

We next argue that if S is a solution, then for all w_i , there exists j such that the solution is disjoint from any (w_i, u_j) arc and any (v_j, w_i) arc. Since the budget is $k = b \cdot h(h - 1)$ we have that: If we cut more than $x_i(h - 1)$ arcs incident to w_i for some $i \in [n]$, then there exists $i' \in [n] \setminus \{i\}$ such that we cut fewer than $x_{i'}(h - 1)$ arcs incident to $w_{i'}$. But this would violate Claim 14. Hence, for any solution S , we can retrieve the assignment of items to bins in the EXACT UNARY BIN PACKING instance \mathcal{I} , by identifying for every $i \in [n]$, the unique value of $j \in [h]$ such that S is disjoint from any (w_i, u_j) arc and any (v_j, w_i) arc and then assigning item x_i to bin J_j . ◀

Besides establishing that UNARY BIN PACKING does not have an FPT algorithm unless $W[1] = \text{FPT}$, Jansen et al. [13] showed that under the stronger assumption² of the Exponential Time Hypothesis (ETH) the well-known $n^{\mathcal{O}(h)}$ -time algorithm is asymptotically almost optimal. The formal statement follows.

► **Proposition 15** ([13]). *There is no algorithm solving the UNARY BIN PACKING problem in $f(h) \cdot n^{o(h/\log h)}$ time for some function f , where h is the number of bins in the input and n is the input length, unless ETH fails.*

² It is known that if ETH is true, then $W[1] \neq \text{FPT}$ [7].

Since our reduction from UNARY BIN PACKING to ESCAD transforms the parameter linearly and the instance size polynomially, we also have a similar ETH based lower bound parameterized by the vertex cover number for ESCAD.

► **Theorem 3.** *There is no algorithm solving ESCAD in $f(k) \cdot n^{o(k/\log k)}$ time for some function f , where k is the vertex cover number of the graph and n is the input length, unless the Exponential Time Hypothesis fails.*

Proof. Follows from the reduction in the proof of Theorem 2 along with Proposition 15. ◀

3.3 NP-hardness of ESCAD on Graphs of Constant Maximum Degree

We show that ESCAD is para-NP-hard when parameterized by the maximum degree.

► **Theorem 4.** *ESCAD is NP-hard in digraphs where each vertex has (in, out) degrees in $\{(1, 6), (6, 1)\}$.*

Proof. We give a polynomial-time reduction from VERTEX COVER on cubic (3-regular) graphs, which is known to be NP-hard [16], to ESCAD. This reduction is a modification of the proof in [16] which shows that DIRECTED FEEDBACK ARC SET is NP-hard. The input to VERTEX COVER consists of a graph G and an integer k ; the task is to decide whether G has a vertex cover of size at most k . Let (G, k) be an instance of VERTEX COVER with n vertices where G is a cubic graph. We construct an ESCAD instance $\mathcal{I}' = (G', k)$ in the following way. The vertex set $V(G') = V(G) \times \{0, 1\}$ and the arc set $A(G')$ is defined by the union of the sets $\{((u, 0), (u, 1)) : u \in V(G)\}$ and $\{((u, 1), (v, 0)) : uv \in E(G)\}$. We call the arcs of the form $((u, 0), (u, 1))$ *internal arcs* and arcs of the form $((u, 1), (v, 0))$ *cross arcs*. Towards the correctness of the reduction, we prove the following claim.

▷ **Claim 16** (★). (G, k) is a yes-instance of VERTEX COVER if and only if (G', k) is a yes-instance of ESCAD.

This shows that ESCAD is NP-hard. Moreover, Since G is a cubic graph, every vertex in D' has (in, out) degree equal to $(1, 6)$ or $(6, 1)$. This completes the proof of Theorem 4. ◀

3.4 Algorithms for ESCAD on Graphs of Bounded Treewidth

Due to Theorem 2, the existence of an FPT algorithm for ESCAD parameterized by various width measures such as treewidth is unlikely. In fact, due to Theorem 3, assuming ETH, even obtaining an algorithm with running time $f(k)n^{o(k/\log k)}$ is not possible, where k is the vertex cover number. On the other hand, this raises a natural algorithmic question – could one obtain an algorithm whose running time matches this lower bound? In this section, we give such an algorithm that is simultaneously, an XP algorithm parameterized by treewidth, an FPT algorithm parameterized by the treewidth and solution size, and also an FPT algorithm parameterized by the treewidth and maximum degree of the input digraph. Moreover, the running time of the algorithm nearly matches the lower bound we have.

Let us note that in the specific case of parameterizing by treewidth and maximum degree, if all we wanted was an FPT algorithm, then we could use Courcelle's theorem at the cost of a suboptimal running time. However, our algorithm in one shot gives us three consequences and as stated earlier, achieves nearly optimal dependence on the treewidth assuming ETH.

Overview of our algorithm. We present a dynamic programming algorithm over tree decompositions. When one attempts to take the standard approach, the main challenge that arises is that by disconnecting strongly connected components, removing an arc can affect vertices far away and hence possibly vertices that have already been forgotten at the current stage of the algorithm. Our solution is to guess the partition of each bag into strongly connected components in the final solution and then keep track of the imbalances of the vertices of the bag under this assumption of components. This allows us to safely forget a vertex as long as its “active” imbalance is zero (any remaining imbalance will be addressed by not strongly connecting the contributing vertices in the future). The remaining difficulty lies in keeping track of how these assumed connections interact with the bag: whether they use vertices already forgotten or those yet to be introduced.

► **Theorem 5.** *An ESCAD instance $\mathcal{I} = (G, k)$ can be solved in time $2^{\mathcal{O}(\text{tw}^2)} \cdot (2\alpha + 1)^{2\text{tw}} \cdot n^{\mathcal{O}(1)}$ where tw is the treewidth of deoriented G , Δ is the maximum degree of G , and $\alpha = \min(k, \Delta)$.*

Since the maximum degree is upper bounded by the instance length (recall footnote in Section 1), this gives an XP algorithm parameterized by treewidth alone. However, when in addition to treewidth we parameterize either by the size of the solution or by the maximum degree this gives an FPT algorithm.

► **Corollary 17.** *ESCAD is FPT parameterized by $\text{tw} + k$, FPT parameterized by $\text{tw} + \Delta$, and XP parameterized by tw alone.*

Recall that in digraphs, multiarcs are permitted. So, we use a variant of the nice tree decomposition notion. This is defined for a digraph G by taking a nice tree decomposition with introduce edge nodes (see Section 2) of the deoriented, simple version of G then expanding each introduce edge node to introduce all parallel copies of arcs one by one. Note that although the new introduce arc nodes introduce *arcs*, the orientation does not affect the decomposition. Let us denote such a tree decomposition of G as $(\mathcal{T}, \{X_t\}_{t \in V(\mathcal{T})})$. Korhonen and Lokshantov [17] gave a $2^{\text{tw}^2} \cdot n^{\mathcal{O}(1)}$ -time algorithm that computes an optimal tree decomposition. Moreover, any tree decomposition can be converted to a nice tree decomposition of the same width with introduce edge nodes in polynomial time [5], and the introduce edge nodes can clearly be expanded to introduce arc nodes in polynomial time. Since the running time of our algorithm dominates the time taken for this step, we may assume that we are given such a tree decomposition. Let G_t be the subgraph of the input graph that contains the vertices and arcs introduced in the subtree rooted at t . We refer to G_t as the past and to all other arcs and vertices as the future.

To tackle ESCAD we need to know whether an arc between vertices in a bag is active in the graph minus a hypothetical solution or not. Towards this, we express the reachability of the graph that lies outside (both past and future) of the current bag as follows.

► **Definition 18.** *For a set X , let (R, ℓ) be a reachability arrangement on X where R is a simple digraph with $V(R) = X$, and ℓ is a labeling $\ell: E(R) \rightarrow \{\text{direct}, \text{past}, \text{future}\}$.*

Let us use $\ell(u, v)$ to denote $\ell((u, v))$. As reachability arrangement implies which vertices of the bag lie in the same strongly connected components we can determine whether an arc is active by checking that its endpoints lie in the same strongly connected component. We aim to track the balance of the vertices in the bag with respect to all past active arcs.

► **Definition 19.** *Given G and R the active imbalance $b_G^R(v)$ of a vertex v in G with respect to R is the imbalance of v in the graph H , i.e. $\deg_H^+(v) - \deg_H^-(v)$, where H is the graph induced on G by the vertices of the strongly connected component of R containing v .*

Although the active imbalance is bounded by Δ , it can be large even when the solution is bounded so we want to instead track how much the active imbalance varies between two graphs.

► **Definition 20.** Given G_1, G_2 , and R the offset imbalance of a vertex v between G_1 and G_2 with respect to R , $\text{off}_{G_1, G_2}^R(v) = b_{G_1}^R(v) - b_{G_2}^R(v)$.

We will consider the offset imbalance between G_t and $G_t - S$ where S is part of a solution. The following lemma allows us to bound this quantity by the size of the solution.

► **Lemma 21** (\star). For each set of arcs $S \subseteq E(G)$, node $t \in V(\mathcal{T})$, simple digraph R on X_t and vertex $v \in X_t$, the offset imbalance of v between $G_t - S$ and G_t with respect to R is between $-|S|$ and $|S|$.

For a solution S we use a suitable reachability arrangement (R, ℓ) , balance labeling B , and part of the solution in the bag W to express a *partial solution*, that is: $S \cap G_t$ along with how vertices of the bag are partitioned into strongly connected components in $G - S$. These give a description of partial solutions that is small enough to guess but detailed enough to admit a dynamic programming approach.

► **Definition 22.** Given a node of the tree decomposition t , a reachability arrangement (R, ℓ) on X_t , a labeling $b: V(R) \rightarrow [-\alpha, \alpha]$, and a subset of arcs $W \subseteq E(G_t[X_t])$ we call a set of arcs $S \subseteq E(G_t)$ compatible with R, ℓ, b, W if all of the following parts hold.

1. S agrees with W on $G_t[X_t]$, that is $S \cap E(G_t[X_t]) = W$.
2. For each arc $e \in \ell^{-1}(\text{direct})$, e is an arc in $G_t[X_t] - S$.
3. For each arc $(u, w) \in \ell^{-1}(\text{past})$ there is a path from u to w in $G_t - S$ that contains no vertices from $X_t \setminus \{u, w\}$ (also called path through the past).
4. For each arc $(u, w) \in \ell^{-1}(\text{future})$ there is no path through the past from u to w (see part 3) and there is a path from u to w in $G - S$ that contains no vertices from $X_t \setminus \{u, w\}$ (also called path through the future).
5. For each vertex $u \in X_t$, the offset imbalance of u between $G_t - S$ and G_t with respect to R is $b(u)$, i.e., $\text{off}_{G_t, G_t - S}^R(u) = b(u)$.
6. For each vertex $u \in V(G_t) \setminus X_t$, the active imbalance of u in $G_t - S$ with respect to $(G_t - S) \cup R$ is zero, i.e., $b_{(G_t - S) \cup R}^{(G_t - S) \cup R}(u) = 0$.

► **Observation 23** (\star). Suppose that S is a solution. For all nodes t there exists R, ℓ, b, W such that $S_1 = S \cap E(G_t)$ is compatible with R, ℓ, b, W .

► **Lemma 24** (\star). Suppose that S is a solution and both $S_1 = S \cap E(G_t)$ and $S_2 \subseteq E(G_t)$ are compatible with R, ℓ, b, W . Then $S' = (S \setminus S_1) \cup S_2$ is also a solution.

The above lemma implies that for fixed t, R, ℓ, b, W all solutions S have the same cardinality of $S \cap G_t$. For fixed t, R, ℓ, b, W to compute existence of some solution S such that $S_1 = S \cap G_t$ is compatible with R, ℓ, b, W , it suffices to compute the minimum cardinality of a subset $S_2 \subseteq E(G_t)$ compatible with R, ℓ, b, W because one can always produce the solution $S' = (S - S_1) \cup S_2$.

Proof of Theorem 5. We will denote by $A[t, R, \ell, b, W]$ the minimum size of an arc subset of G_t that is compatible with R, ℓ, b , and W . In our decomposition $(\mathcal{T}, \{X_t\}_{t \in V(\mathcal{T})})$ the root node r has $X_r = \emptyset$ and $G_r = G$ so $A[r, \emptyset, \emptyset, \emptyset, \emptyset]$ is equal to the minimum size of a solution. In order to compute $A[r, \emptyset, \emptyset, \emptyset, \emptyset]$ we employ the standard bottom up dynamic programming over treewidth decomposition approach.

4:14 On the Parameterized Complexity of Eulerian Strong Component Arc Deletion

For leaf nodes $X_t = \emptyset$, hence, the graphs and labelings are also empty and the empty arc set is vacuously compatible with them $A[t, \emptyset, \emptyset, \emptyset, \emptyset] = 0$.

For every non-leaf node t and graph R on X_t we first calculate the strongly connected components of R . Then we can calculate the active imbalance $b_{G_t}^R(v)$ of each vertex $v \in X_t$ in G_t with respect to R . Then for each ℓ , b , and W we calculate $A[t, R, \ell, b, W]$ based on the type of the node t . We give only an informal description here, the full formulae and proofs can be found in the full version.

Introduce vertex node: When t is an introduce vertex node and its child is t' with $X_t = X_{t'} \cup \{v\}$ we know that v will be isolated in G_t so we can discount any reachability arrangements where there are direct or past arcs incident to v . Additionally, the active imbalance on v must be zero. Any new future connections should be reflected in the old reachability arrangement, that is, if the new arrangement contains a future arc from u to v and from v to w there should be a future arc between u and w in the old arrangement. No arcs were introduced or forgotten so the set W remains the same.

Introduce arc node: Assume t introduces arc (u, v) and its child is t' . In any case, if u and v are in different strongly connected components then the new arc is inactive so it does not influence active degrees. We recognize two distinct cases based on whether this new arc belongs to S . On one hand, say the new arc $(u, v) \notin S$, then it may realize a future path from u to v . Also, if u and v are in the same strongly connected component, then the added (u, v) arc changes the active imbalance of u and v by one in G_t but also in $G_t - S$ so the offset imbalance remains the same. On the other hand, if $(u, v) \in S$, then the active degree of its endpoints changes in G_t but it does not change in $G_t - S$, hence, the offset imbalance changes by one. Note that the introduced arc (u, v) may be one among multiple parallel copies of a multiarc – the only minor difference if we did not allow multiarcs would be to not allow the label on (u, v) in t' to be direct.

Forget node: If t is a forget node with child t' such that $X_t = X_{t'} \setminus \{v\}$ then we need to ensure that the forgotten vertex has zero active imbalance in $G_t - S$ and that there are no future arcs incident to it in the old arrangement. Zero active imbalance is equivalent to an offset imbalance of $-b_{G_t}^R(v)$, which we have precalculated. Also, the only change to the remaining reachability arrangement should be new past arcs where there was previously a path through v .

Join node: When merging two nodes t_1 and t_2 to a parent join node t the reachability arrangements should be nearly the same. The notable exception is that past arcs in the parent arrangement can be either past in both child arrangements or we can have past arc in one arrangement while there is a future arc on the other arrangement. In a similar way, we need to consider for each $u \in X_t$ how the imbalance $b(u)$ in G_t is made up of parts in G_{t_1} and G_{t_2} . The new compatible solutions are unions of the solutions compatible with pairs of such arrangements. Their overlap is exactly W so the size of the union is simply the sum of their sizes minus $|W|$.

For a fixed node t there are 4^{tw^2} reachability arrangements on X_t , $(2\alpha + 1)^{tw}$ possible b 's, and 2^{tw^2} possible W 's. Both introduce vertex and introduce arc node compute their entry from a fixed entry of their child node in $n^{\mathcal{O}(1)}$ time. Forget node is computed in $2^{tw} \cdot 4^{tw} \cdot n^{\mathcal{O}(1)}$ while join node is computed in $3^{tw^2} \cdot (2\alpha + 1)^{tw} \cdot n^{\mathcal{O}(1)}$ time.

It is known that the total number of nodes in the nice tree decomposition with introduce arc nodes is $n^{\mathcal{O}(1)}$ and it can be observed that this still holds for the extension on multiarcs. Hence, the overall run time is

$$(4^{tw^2} \cdot (2\alpha + 1)^{tw} \cdot 2^{tw^2}) \cdot (2^{tw} \cdot 4^{tw} + 3^{tw^2} \cdot (2\alpha + 1)^{tw}) \cdot n^{\mathcal{O}(1)} = 24^{tw^2} \cdot (2\alpha + 1)^{2tw} \cdot n^{\mathcal{O}(1)}. \blacktriangleleft$$

4 Our Results for ESCAD on Simple Digraphs

In this section, we study ESCAD on simple digraphs, which we formally define as follows.

SIMPLE EULERIAN STRONG COMPONENT ARC DELETION (SESCAD)

Input: A simple digraph G , an integer k

Question: Is there a subset $R \subseteq E(G)$ of size $|R| \leq k$ such that in $G - R$ each strongly connected component is Eulerian?

Let us begin by stating a simple observation that enables us to make various inferences regarding the complexity of SESCAD based on the results we have proved for ESCAD.

► **Observation 25.** *Consider an ESCAD instance $\mathcal{I} = (G, k)$. If we subdivide every arc (u, v) into $(u, w), (w, v)$ (using a new vertex w) then we get an equivalent SESCAD instance $\mathcal{I}' = (G', k)$ with $|V(G')| = |V(G)| + |E(G)|$ and $|E(G')| = 2|E(G)|$. Moreover, each arc of the solution to \mathcal{I} is mapped to one respective arc of the subdivision and vice versa.*

4.1 Hardness Results for SESCAD

We first discuss the implications of Theorem 1, Theorem 2 and Theorem 4 for SESCAD along with Observation 25.

► **Corollary 26.** *SESCAD is $W[1]$ -hard when parameterized by the solution size.*

Proof. Follows from Theorem 1 and Observation 25. ◀

► **Observation 27.** *If we subdivide all arcs in a digraph G that has a vertex cover X , we get a simple digraph G' such that $G' - X$ is the disjoint union of directed stars.*

► **Corollary 28.** *SESCAD is $W[1]$ -hard parameterized by minimum modulator size to disjoint union of directed stars.*

Using the stronger assumption of ETH, we have the following result.

► **Theorem 29.** *There is no algorithm solving SESCAD in $f(k) \cdot n^{o(k/\log k)}$ time for some function f , where k is the size of the smallest vertex set that must be deleted from the input graph to obtain a disjoint union of directed stars and n is the input length, unless the Exponential Time Hypothesis fails.*

Proof. The reduction in the proof of Theorem 2 along with Proposition 15, Observation 25 and Observation 27 implies the statement. ◀

Note that the above result rules out an FPT algorithm for SESCAD parameterized by various width measures such as treewidth and even treedepth.

► **Theorem 30.** *SESCAD is NP-hard in simple digraphs where each vertex has (in, out) degrees in $\{(1, 1), (1, 6), (6, 1)\}$.*

Proof. Follows from Theorem 4 and Observation 25. ◀

4.2 FPT Algorithms for SESCAD

Firstly, the FPT algorithms discussed in the previous section naturally extend to SESCAD. However, for SESCAD, the lower bound parameterized by modulator to a disjoint union of directed stars leaves open the question of parameterizing by larger parameters. For instance, the vertex cover number.

To address this gap, we provide an FPT algorithm for SESCAD parameterized by *vertex integrity*, a parameter introduced by Barefoot et al. [1].

► **Definition 31** (Vertex Integrity). An undirected graph $G = (V, E)$ has *vertex integrity* k if there exists a set of vertices $M \subseteq V$, called a k -separator, of size at most k such that when removed each connected component has size at most $k - |M|$. A directed graph has vertex integrity k if and only if the underlying undirected graph has vertex integrity k . The notion of a k -separator in digraphs carries over naturally from the undirected setting.

FPT algorithms parameterized by vertex integrity have gained popularity in recent years due to the fact that several problems known to be $W[1]$ -hard parameterized even by treedepth can be shown to be FPT when parameterized by the vertex integrity [10]. Since Corollary 28 rules out FPT algorithms for SESCAD parameterized by treedepth, it is natural to explore SESCAD parameterized by vertex integrity and our positive result thus adds SESCAD to the extensive list of problems displaying this behavior.

Moreover, this FPT algorithm parameterized by vertex integrity implies that SESCAD is also FPT when parameterized by the vertex cover number and shows that our reduction for ESCAD parameterized by the vertex cover number requires multiarcs for fundamental reasons and cannot be just adapted to simple digraphs with more work.

We will use as a subroutine the well-known FPT algorithm for ILP-FEASIBILITY. The ILP-FEASIBILITY problem is defined as follows. The input is a matrix $A \in \mathbb{Z}^{m \times p}$ and a vector $b \in \mathbb{Z}^{m \times 1}$ and the objective is to find a vector $\bar{x} \in \mathbb{Z}^{p \times 1}$ satisfying the m inequalities given by A , that is, $A \cdot \bar{x} \leq b$, or decide that such a vector does not exist.

► **Proposition 32** ([14, 15, 9]). ILP-FEASIBILITY can be solved using $\mathcal{O}(k^{2.5k+o(k)} \cdot L)$ arithmetic operations and space polynomial in L , where L is the number of bits in the input and k is the number of variables.

► **Theorem 6.** ESCAD on simple digraphs is FPT parameterized by the vertex integrity of the graph.

Proof. Consider an instance (G, p) of SESCAD, where G has vertex integrity at most k . Suppose that this is a yes-instance with a solution S and let M be a k -separator of G . Without loss of generality, assume that $V(G) = [n]$ and $M = [|M|]$. In our algorithm, we only require the fact that since M is a k -separator in a digraph G , every weakly connected component of $G - M$ has size at most k (recall, the definition of vertex integrity bounds the component sizes even more). Further, we remark that our algorithm does not require a k -separator to be given as input since there is an FPT algorithm parameterized by k to compute it [8].

We next guess those arcs of S that have both endpoints in M , remove them and adjust p accordingly. The number of possible guesses is $2^{\mathcal{O}(k^2)}$. Henceforth, we assume that every arc in the hypothetical solution S has at least one endpoint disjoint from M .

We next guess the reachability relations between the vertices of M in $G - S$. The correct guess is called the *reachability signature* of M in $G - S$, denoted by σ , which is a set of ordered pairs where, for every $m_1, m_2 \in M$, $(m_1, m_2) \in \sigma$ if and only if m_2 is reachable from m_1 in $G - S$. The number of possibilities for σ is clearly bounded by $2^{\mathcal{O}(k^2)}$.

For every simple digraph comprised of at most $|M| + k$ vertices and every possible injective mapping λ of M to the vertices of this digraph, we define the *type* of this digraph as the label-preserving isomorphism class with the labeling λ . Denote the set of all types by **Types**. For each type $\tau \in \mathbf{Types}$, we denote by G_τ a fixed graph of this type that we can compute in time depending only on k . Due to the labeling injectively mapping M to the vertices of G_τ , we may assume that $M \subseteq V(G_\tau)$.

The number of types is clearly bounded by a function of k and for each weakly connected component (from now onwards, simply called a component) C of $G - M$ and graph $G_C = G[C \cup M]$ with the vertices of M mapped to themselves by the identity labeling on M , denoted λ_M , we compute the type of the graph G_C . From now on, we drop the explicit reference to λ_M as it will be implied whenever we are handling the graph G_C . For every type τ , we also compute the number n_τ of components C such that G_C is of type τ . Since the type of each G_C can be computed in $f(k)$ -time for some function f , this step takes FPT time.

Following that, for every component C , and every arc set S_C in G_C , we check whether the type of $G_C - S_C$ (with labeling λ_M) is *compatible* with σ . To be precise, for a set S_C of arcs in the digraph G_C , we verify that every vertex of C is balanced in its strongly connected component in the graph $G'_C = G_C - S_C + \sigma$. If the answer to this check is yes, then this is a compatible type. Notice that by adding the ordered pairs in σ as arcs to $G_C - S_C$, we ensure that the arcs of the graph we take into account in this check on balances of the vertices in C (i.e., active arcs) are exactly all those arcs that are already in $\text{strong}(G_C - S_C)$ plus those arcs of $G_C - S_C$ that *would* be inside a strongly connected component *if* the relations in σ were realized. Since each component C has size bounded by k , the number of possibilities for S_C is bounded by a function of k for each component (here, we crucially use the fact that we have a simple digraph), and hence, in FPT time, we can compute a table Γ stating, for every C and S_C subset of arcs in G_C , whether the type of $G_C - S_C$ is compatible with σ .

Notice that for each component, deleting the arcs of the hypothetical solution S from each component C transitions G_C from one type to another type that is compatible with σ . To be precise, for each C and set $S_C = S \cap A(G_C)$, we can think of S_C as taking G_C from the type of G_C (call it τ_1) to the type of $G_C - S_C$ (call it τ_2), at cost $|S_C|$. Moreover, the type τ_2 is compatible with σ . Thus, the table Γ encodes the cost of transitioning each graph G_C to a type compatible with σ . This can be expressed by a value $\text{cost}(\tau_1, \tau_2)$ for every pair of types. If τ_2 is not compatible with σ , then set this value to be prohibitively high, say the number of arcs in G plus one. Otherwise, $\text{cost}(\tau_1, \tau_2)$ is given by the table Γ .

In our next step, we guess a set of $\mathcal{O}(k^2)$ types such that for every pair of vertices $m_1, m_2 \in M$, if σ requires that m_1 can reach m_2 , then there is a sequence of vertices of M starting at m_1 and ending in m_2 such that for every consecutive ordered pair (x, y) in this sequence, either (x, y) is an arc in $G[M]$ (and since it is not already deleted, it is disjoint from S) or there is an x - y path with all internal vertices through a subgraph that belongs to one of these $\mathcal{O}(k^2)$ types. Call this set of types T^* . The bound on the size of T^* comes from the fact that there are $\mathcal{O}(k^2)$ pairs in σ .

Finally, whether or not the vertices of M are balanced in $\text{strong}(G - S)$ is determined entirely by the number of graphs of each type in $G - S$ subject to the types in T^* occurring. So, for every type, we determine the imbalance imposed by the type on each vertex of M (taking σ into account). To be precise, for every type τ and vertex $u \in M$, the imbalance on u due to τ is denoted by $I(\tau, u)$ and is obtained by subtracting the number of active incoming arcs on u from the number of active outgoing arcs on u , where an arc $(p, q) \in A(G_\tau)$ where $u \in p, q$ is active, if and only if it lies in the same strongly connected component as u in the graph $G_\tau + \sigma$.

4:18 On the Parameterized Complexity of Eulerian Strong Component Arc Deletion

All of the above requirements can be formulated as an ILP-FEASIBILITY instance with $f(k)$ variables that effectively minimizes the total costs of all the required type transitions. More precisely, for every pair of types τ_1 and τ_2 , we have a variable x_{τ_1, τ_2} that is intended to express the number of graphs G_C of type τ_1 that transition to type τ_2 . We only need to consider variables x_{τ_1, τ_2} where τ_1 is the type of some G_C and τ_2 is compatible with σ . So, we restrict our variable set to this. Moreover, for every τ that is compatible with σ , we have a variable y_τ that is intended to express the number of components C such that G_C transitions to type τ .

Then, we have constraints that express the following:

1. The cost of all the type transitions is at most p .

$$\sum_{\tau_1, \tau_2 \in \text{Types}} \text{cost}(\tau_1, \tau_2) \cdot x_{\tau_1, \tau_2} \leq p$$

2. For each type τ in T^* , there is at least one transition to τ . This will ensure that the reachability relations required by σ are achieved.

$$\sum_{\tau_1 \in \text{Types}} x_{\tau_1, \tau} \geq 1$$

3. For every component C , G_C transitions to some type compatible with σ . So, for every type τ , we have:

$$\sum_{\tau_2 \in \text{Types}} x_{\tau, \tau_2} = n_\tau$$

Recall that n_τ denotes the number of components C such that G_C is of type τ and we have computed it already.

4. The number of components C such that G_C transitions to type τ , is given by summing up the values of $x_{\tau_1, \tau}$ over all possible values of τ_1 .

$$\sum_{\tau_1 \in \text{Types}} x_{\tau_1, \tau} = y_\tau$$

5. The total imbalance imposed on each vertex of M by the existing arcs incident to it, plus the imbalance imposed on it by the types to which we transition, adds up to 0.

For each $u \in M$, let ρ_u denote the imbalance on u imposed by those arcs of $G[M]$ that are incident to u and active in the graph $G[M] + \sigma$. The imbalance imposed on u by a particular type τ is $I(\tau, u)$ and this needs to be multiplied by the number of “occurrences” of this type after removing the solution, i.e., the value of y_τ .

Hence, we have the following constraint for every $u \in M$.

$$\rho_u + \sum_{\tau \in \text{Types}} I(\tau, u) y_\tau = 0$$

6. Finally, we need the variables to all get non-negative values. So, for every $\tau_1, \tau_2 \in \text{Types}$, we add $x_{\tau_1, \tau_2} \geq 0$ and for every $\tau \in \text{Types}$, $y_\tau \geq 0$.

It is straightforward to convert the above constraints into the form of an instance of ILP-FEASIBILITY. Since the number of variables is a function of k , Proposition 32 can be used to decide feasibility in FPT time. From a solution to the ILP-FEASIBILITY instance, it is also straightforward to recover a solution to our instance by using the table Γ . ◀

5 Conclusions

We have resolved the open problem of Cechlárová and Schlotter [3] on the parameterized complexity of the Eulerian Strong Component Arc Deletion problem by showing that it is $W[1]$ -hard and accompanied it with further hardness results parameterized by the vertex cover number and max-degree of the graph. On the positive side, we showed that though the problem is inherently difficult in general, certain combined parameterizations (such as treewidth plus either max-degree or solution size) offer a way to obtain FPT algorithms.

Our work points to several natural future directions of research on this problem.

1. Design of (FPT) approximation algorithms for ESCAD?
2. ESCAD parameterized by the solution size is FPT on tournaments [4]. For which other graph classes is the problem FPT by the same parameter?
3. Our FPT algorithm for SESCAD parameterized by vertex integrity is only aimed at being a characterization result and we have not attempted to optimize the parameter dependence. So, a natural follow up question is to obtain an algorithm that is as close to optimal as possible.
4. Are there parameterizations upper bounding the solution size, for which ESCAD is FPT? For instance, the size of the minimum directed feedback arc set of the input digraph. Notice that in the reduction of Theorem 1, we obtain instances with unboundedly large minimum directed feedback arc sets due to the imbalance gadgets starting at the vertex s_j for some color class j and ending at the vertices in $\{x_u \mid u \in \text{color class } j\}$.

References

- 1 C. A. Barefoot, R. Entringer, and H. C. Swart. Vulnerability in graphs—a comparative survey. *JCMCC*, 1:13–22, 1987.
- 2 Václav Blažej, Satyabrata Jana, M. S. Ramanujan, and Peter Strulo. On the parameterized complexity of eulerian strong component arc deletion. *CoRR*, abs/2408.13819, 2024. doi:10.48550/arXiv.2408.13819.
- 3 Katarína Cechlárová and Ildikó Schlotter. Computing the deficiency of housing markets with duplicate houses. In Venkatesh Raman and Saket Saurabh, editors, *Parameterized and Exact Computation - 5th International Symposium, IPEC 2010, Chennai, India, December 13-15, 2010. Proceedings*, volume 6478 of *Lecture Notes in Computer Science*, pages 72–83. Springer, 2010. doi:10.1007/978-3-642-17493-3_9.
- 4 Robert Crowston, Gregory Z. Gutin, Mark Jones, and Anders Yeo. Parameterized eulerian strong component arc deletion problem on tournaments. *Inf. Process. Lett.*, 112(6):249–251, 2012. doi:10.1016/j.ipl.2011.11.014.
- 5 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
- 6 Marek Cygan, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Ildikó Schlotter. Parameterized complexity of eulerian deletion problems. *Algorithmica*, 68(1):41–61, 2014. doi:10.1007/S00453-012-9667-X.
- 7 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.
- 8 Pål Grønås Drange, Markus S. Dregi, and Pim van 't Hof. On the computational complexity of vertex integrity and component order connectivity. *Algorithmica*, 76(4):1181–1202, 2016. doi:10.1007/S00453-016-0127-X.
- 9 András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Comb.*, 7(1):49–65, 1987. doi:10.1007/BF02579200.

- 10 Tatsuya Gima, Tesshu Hanaka, Masashi Kiyomi, Yasuaki Kobayashi, and Yota Otachi. Exploring the gap between treedepth and vertex cover through vertex integrity. *Theor. Comput. Sci.*, 918:60–76, 2022. doi:10.1016/J.TCS.2022.03.021.
- 11 Alexander Göke, Dániel Marx, and Matthias Mnich. Parameterized algorithms for generalizations of directed feedback vertex set. *Discret. Optim.*, 46:100740, 2022. doi:10.1016/J.DISOPT.2022.100740.
- 12 Prachi Goyal, Pranabendu Misra, Fahad Panolan, Geevarghese Philip, and Saket Saurabh. Finding even subgraphs even faster. *J. Comput. Syst. Sci.*, 97:1–13, 2018. doi:10.1016/J.JCSS.2018.03.001.
- 13 Klaus Jansen, Stefan Kratsch, Dániel Marx, and Ildikó Schlotter. Bin packing with fixed number of bins revisited. *J. Comput. Syst. Sci.*, 79(1):39–49, 2013. doi:10.1016/J.JCSS.2012.04.004.
- 14 Hendrik W. Lenstra Jr. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983. doi:10.1287/MOOR.8.4.538.
- 15 Ravi Kannan. Minkowski’s convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, 1987. doi:10.1287/MOOR.12.3.415.
- 16 Richard M. Karp. Reducibility among combinatorial problems. In Michael Jünger, Thomas M. Lieblich, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, editors, *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*, pages 219–241. Springer, 2010. doi:10.1007/978-3-540-68279-0_8.
- 17 Tuukka Korhonen and Daniel Lokshantov. An improved parameterized algorithm for treewidth. In *STOC*, pages 528–541. ACM, 2023. doi:10.1145/3564246.3585245.

Unsplittable Flow on a Short Path

Ilan Doron-Arad ✉

Computer Science Department, Technion, Haifa, Israel

Fabrizio Grandoni ✉

IDSIA, USI-SUPSI, Lugano, Switzerland

Ariel Kulik ✉ 

Computer Science Department, Technion, Haifa, Israel

Abstract

In the Unsplittable Flow on a Path problem (UFP), we are given a path graph with edge capacities and a collection of tasks. Each task is characterized by a demand, a profit, and a subpath. Our goal is to select a maximum profit subset of tasks such that the total demand of the selected tasks that use each edge e is at most the capacity of e . BagUFP is the generalization of UFP where tasks are partitioned into bags, and we are allowed to select at most one task per bag. UFP admits a PTAS [Grandoni,Mömke,Wiese'22] but not an EPTAS [Wiese'17]. BagUFP is APX-hard [Spektsma'99] and the current best approximation is $O(\log n / \log \log n)$ [Grandoni,Ingala,Uniyal'15], where n is the number of tasks.

In this paper, we study the mentioned two problems when parameterized by the number m of edges in the graph, with the goal of designing faster parameterized approximation algorithms. We present a parameterized EPTAS for BagUFP, and a substantially faster parameterized EPTAS for UFP (which is an FPTAS for $m = O(1)$). We also show that a parameterized FPTAS for UFP (hence for BagUFP) does not exist, therefore our results are qualitatively tight.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Knapsack, Approximation Schemes, Parameterized Approximations

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.5

Related Version *Full Version*: <https://arxiv.org/abs/2407.10138> [23]

Funding *Fabrizio Grandoni*: Partially supported by the SNSF Grant 200021_2000731/1.

Ariel Kulik: This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 852780-ERC (SUBMODULAR).

1 Introduction

In the classical Unsplittable Flow on a Path problem (UFP) we are given an m -edge path graph $G = (V, E)$ with (non-negative integer) edge capacities $u : E \rightarrow \mathbb{N}$, and a collection of n tasks T . Each task i is characterized by a demand $d(i) \in \mathbb{N}$, a weight (or profit) $w(i) \in \mathbb{N}$, and a subpath $P(i)$ ¹. A feasible solution consists of a subset of (selected) tasks $S \subseteq T$ such that, for each edge e , $\sum_{i \in S: e \in P(i)} d(i) \leq u(e)$. In other words, the total demand of the selected tasks using each edge e cannot exceed the capacity of e . Our goal is to compute a feasible solution OPT of maximum total profit $\text{opt} = w(\text{OPT}) := \sum_{i \in \text{OPT}} w(i)$.

UFP has several direct and indirect applications [7, 8, 11, 12, 16, 17, 18, 21, 41, 43]. For example, one might interpret G as a time interval subdivided into time slots (the edges). At each time slot we are given some amount of a considered resource, say, energy. The tasks

¹ Throughout this paper, for a subpath P , we sometimes use P also to denote the corresponding set of edges $E(P)$: the meaning will be clear from the context.



represent jobs that we might execute, therefore gaining a profit. However each executed job will consume some amount of the shared resource during its execution, thus we might not be able to execute all the jobs (hence we need to perform a selection).

UFP is strongly NP-hard [11, 18] and it is well-studied in terms of approximation algorithms. After a long sequence of improvements [2, 3, 5, 6, 9, 11, 14, 15, 34, 37, 38, 36], a PTAS for UFP was eventually achieved by Grandoni, Mömke and Wiese [35]. We recall that a PTAS (for a maximization problem) is an algorithm parameterized by $\varepsilon > 0$, which provides a $(1 - \varepsilon)$ approximation in time $|I|^{O_\varepsilon(1)}$, where $|I|$ is the input size. EPTASs and FPTASs are defined similarly, however with running times of the form $f(1/\varepsilon) \cdot |I|^{O(1)}$ and $(|I|/\varepsilon)^{O(1)}$, resp., where $f(\cdot)$ is a computable function. Wiese [49] proved that UFP, parameterized by the number of selected tasks, is $W[1]$ -hard: this excludes the existence of an EPTAS for UFP by standard reductions (unless $FPT=W[1]$ [42]).

In the above scenario there is no flexibility on the time when a job is executed. The **BagUFP** problem is a generalization of UFP which was introduced to allow for such flexibility. Here we are given the same input as UFP, plus a partition of the tasks into ℓ bags $\mathcal{B} = \{B_1, \dots, B_\ell\}$, $\dot{\cup}_{j=1}^\ell B_j = T$. A feasible solution S has to satisfy the capacity constraints as in UFP, plus the extra constraint that at most one task per bag can be selected, namely $|S \cap B_j| \leq 1$ for $j = 1, \dots, \ell$. This easily captures jobs that can be executed at different times (and even more general settings). For example, if a job can be executed within a given time window (also known as the time-windows UFP problem), it is sufficient to create a bag that contains multiple copies of the same task which differ only in the subpath $P(i)$ (with one subpath per potential valid scheduling time). **BagUFP** is APX-hard [47], which rules out the existence of a PTAS for it. The current best approximation ratio for **BagUFP** is $O(\log n / \log \log n)$ [33], slightly improving on the $O(\log n)$ approximation in [13]. A constant approximation for **BagUFP** is known for the cardinality version of the problem [33], i.e. when all the profits are 1. Bag constraints are frequently added to other classic optimization problems, such as makespan minimization [22, 32], knapsack [46, 44, 40], and bin packing [24, 25, 31].

1.1 Our Results and Techniques

The mentioned PTAS for UFP [35] has a very poor dependence on ε in the running time, which makes it most likely impractical. Though an improvement of the running time is certainly possible, as mentioned before an EPTAS for UFP does not exist (unless $FPT = W[1]$). The situation for **BagUFP** is even worse: here even a PTAS does not exist (unless $P = NP$), and currently finding a constant approximation algorithm (which might exist) is a challenging open problem.

Motivated by the above situation, it makes sense to consider parameterized approximation algorithms for UFP and **BagUFP**. The general goal here is to identify some integer parameter p that captures some relevant aspect of the input (or some property of the output), and try to design approximation algorithms whose running time is better than the state of the art when p is sufficiently small. In particular a parameterized PTAS (**p**-PTAS) is defined similarly to a PTAS, however with running time of the form $f(p)|I|^{O_\varepsilon(1)}$ for some commutable function $f(\cdot)$. Parameterized EPTAS (**p**-EPTAS) and parameterized FPTAS (**p**-FPTAS) are defined similarly, w.r.t. EPTAS and FPTAS resp. More explicitly, a **p**-EPTAS has a running time of the form $f(p + 1/\varepsilon)|I|^{O(1)}$, while a **p**-FPTAS has a running time of the form $f(p)(|I|/\varepsilon)^{O(1)}$. For a meaningful choice of p , it makes sense to search for a **p**-EPTAS (or better) for UFP, and for a **p**-PTAS (or better) for **BagUFP**.

Probably the most standard parameter is the number k of tasks in the desired solution. This is also the objective function for the cardinality version of the problems (with profits equal to 1). Wiese [49] proved that UFP is $W[1]$ -hard under this parametrization, which

rules out a p-EPTAS. He also presented a p-PTAS for the cardinality version of UFP with parameter k (later improved by the PTAS in [35], which also works for arbitrary profits). To the best of our knowledge, the same parametrization of BagUFP was not studied in the literature.

In this paper we focus on the parameter m , namely the number of edges in G - the length of the path. This makes sense in the realistic scenarios where $n \gg m$ i.e., there are significantly more jobs than time slots. For example, such UFP instances occur in personnel scheduling [48, 4, 10, 1] where, e.g., workers are assigned to shifts within a working day ($m \approx 8$ working hours), or for an interval of days in the week ($m = 7$ days). We achieve the following main results:

1.1.1 Algorithms and Hardness for BagUFP

A simple reduction from Partition shows that (assuming $P \neq NP$) there is no FPTAS for BagUFP even for $m = 2$ (for $m = 1$ an FPTAS exists since the problem is equivalent to Multiple Choice Knapsack). As an obvious corollary, there is no p-FPTAS with parameter m for the same problem (see Section 4).

► **Theorem 1.** *Unless $P = NP$, there is no FPTAS for BagUFP even in the case $m = 2$.*

► **Corollary 2.** *Unless $P = NP$, there is no p-FPTAS for BagUFP parametrized by the path length m .*

Hence, qualitatively speaking, the best one can hope for is a p-EPTAS. This is precisely what we achieve (see Section 2).

► **Theorem 3.** *There is a p-EPTAS for BagUFP parametrized by the path length m . Its running time is $2^{(m/\varepsilon^{1/\varepsilon})^{O(1)}} \cdot |I|^{O(1)}$.*

Our approach substantially differs from previous algorithmic approaches for UFP (see, e.g., [35] and references therein) which relied on concepts such as classification of items by *demands* and probabilistic arguments. We observe that the bag constraints induce a matroid (more specifically, a partition matroid with capacity 1 for each set). Therefore we consider the standard LP relaxation for a partition matroid (which has integral basic solutions), and augment it with the m linear constraints corresponding to the capacity constraints. As proved in [39], a basic optimal solution x^* to this LP (which can be computed in polynomial time for arbitrary m) has at most $2m$ fractional values (with value strictly between 0 and 1). The variables with value 1 in x^* induce a feasible BagUFP solution with profit at least the optimal LP profit minus (almost) the profit of $2m$ tasks: this is problematic if the latter tasks have a profit comparable to $\text{opt} = w(\text{OPT})$, where OPT is some reference optimal solution.

We can avoid the above issue as follows. Let H be the (*heavy*) tasks with profit at least $\frac{\varepsilon}{m} \cdot \text{opt}$. We can guess the heavy tasks $H \cap \text{OPT}$ in OPT (which are at most m/ε many), reduce the problem (i.e., remove all the tasks in the bags containing tasks from $H \cap \text{OPT}$, remove tasks in H , and reduce the available capacity of every edge by the total demand of $\text{OPT} \cap H$ for the specific edge), and apply the mentioned LP-rounding technique to the remaining (*light*) tasks. Now the drop of the fractional variables reduces the profit by at most $2\varepsilon \cdot \text{opt}$, leading to a $1 + O(\varepsilon)$ approximation. Unfortunately, this algorithm would take time $|H|^{\Omega(m/\varepsilon)}$, which is still not compatible with a p-EPTAS.

In order to circumvent the latter issue, we exploit the notion of representative sets, which was introduced in [26, 27, 28] to deal with a class of maximization problems with a single budget constraint. In contrast, we construct a representative set in the more general regime

of multiple budget constraints imposed by the unsplittable flow setting. In more detail, in \mathbf{p} -EPTAS time, we are able to compute a (representative) subset of tasks R of size depending only on m and $1/\varepsilon$, such that there exists a nearly optimal solution S such that $S \cap H \subseteq R$. Therefore, one can restrict to R the above guessing of heavy tasks, which takes $|R|^{O(m/\varepsilon)}$ time: this is now compatible with a \mathbf{p} -EPTAS. We remark that our techniques, combined with the representative set techniques of [26, 27, 28], can give a \mathbf{p} -EPTAS for the more general problem of UFP with a general matroid constraint. We leave such efforts to the journal version of the paper. On the other hand, UFP with a general matroid is somewhat harder since an FPTAS is ruled out even for an instance with path of length 1 (a single budget constraint) [30], and an FPTAS exists only for laminar matroids [29].

1.1.2 Algorithms and Hardness for UFP

We start by showing that there is no \mathbf{p} -FPTAS for UFP parameterized by m . This, together with Theorem 3, gives a tight bound for UFP in the short path point-of-view. Notice that this is not implied by Theorem 1 since UFP is a special case of \mathbf{BagUFP} .

► **Theorem 4.** *Unless $\mathbf{FPT}=\mathbf{W}[1]$, there is no \mathbf{p} -FPTAS for UFP parametrized by the path length m .*

Unlike previous hardness results [11, 18, 47, 49, 19] for UFP and its variant, which rely on a path of polynomial length in the input size, our lower bound requires having UFP instances with a *short* path. Namely, the number of tasks is significantly larger than the length of the path. Our starting point is to obtain a hardness result for a *multiple choice* variant of k -subset sum in which the numbers are partitioned into sets A_1, \dots, A_k , each set with n numbers, and the goal is to select one number from each set such their sum is exactly a given target value. We use color-coding to show that multiple-choice k -subset sum does not have an FPT-algorithm unless $\mathbf{W}[1]=\mathbf{FPT}$ (which may be useful for other hardness results). Then, we reduce multiple-choice k -subset sum to UFP by constructing a UFP instance with $m = O(k)$ edges and with polynomial weights. Roughly, we interpret the edges of the path in correspondence to the k sets A_1, \dots, A_k . The constructed instance has a pair of tasks z_j^i, q_j^i , with complementary subpaths, for every number $j = 1, \dots, n$ in the i -th set A_i . Along with a carefully defined demand and weight functions, This UFP instance satisfies that exactly k pairs can be chosen for a sufficiently high weight if and only if the original subset sum instance has a solution. We remark that this construction utilizes the short path in a non-trivial manner. Due to space constraints, the proof is given in the full version of the paper [23].

Theorem 3 already provides a \mathbf{p} -EPTAS for UFP. We are however able to derive a \mathbf{p} -EPTAS with a substantially better running time (see Section 3).

► **Theorem 5.** *There is an \mathbf{p} -EPTAS for UFP parameterized by the path length m , with running time $O\left(\frac{n^3}{\varepsilon} + \left(\frac{1}{\varepsilon}\right)^{O(m^2)} m^3 \log n\right)$.*

In particular, for $m \leq C \cdot \sqrt{\log_{\frac{1}{\varepsilon}} n}$, for a sufficiently small constant $C > 0$, our running time is the running time of an FPTAS. We recall that achieving an FPTAS (or even an EPTAS) for UFP in general is not possible and the previous state of the art for UFP with a constant number of edges is the PTAS for the general problem [35].

The basic idea of the algorithm is as follows. Consider all the tasks T_φ whose path is φ . Let opt_φ be the profit of some optimal solution OPT restricted to T_φ , i.e. $\text{opt}_\varphi = w(\text{OPT} \cap T_\varphi)$. Given the value of opt_φ , it is sufficient to find a minimum-demand subset of tasks $S_\varphi \subseteq T_\varphi$

with profit at least opt_φ : the union of the sets S_φ would be feasible and optimal. To achieve the target running time we use this basic idea along with rounding of the weights and a coarse guessing of the values opt_φ . By a standard rounding argument, we can assume that the weights are in $[\frac{n}{\varepsilon}]$ while loosing a factor $1 - \varepsilon$ in the approximation. This allows us to pre-compute the minimal demand subset of T_φ which attains a threshold rounded weight, for every possible threshold, using a standard dynamic program. The pre-computed subsets are used to reconstruct a solution S_φ from the value of opt_φ . Finally, we guess the values of opt_φ up to an additive error of $\approx \frac{\varepsilon}{m^2} \cdot w(\text{OPT})$. This coarse guess of the values of opt_φ allows us to enumerate over all possible guesses within the running time, while only introducing an additional $1 - \varepsilon$ factor in the approximation.

1.2 Preliminaries

For every $n \in \mathbb{N}$ we use $[n] = \{1, \dots, n\}$. We use $(G, u, T, P, d, w, \mathcal{B})$ to denote a BagUFP instance and by (G, u, T, P, d, w) to denote a UFP instance. Given an instance I of UFP or BagUFP, we let $\text{OPT}(I)$ denote some reference optimal solution, and $\text{opt}(I) = w(\text{OPT}(I))$ be its profit. We use $|I|$ to denote the encoding size of I . When I is clear from the context, we simply use OPT and opt , resp. Given a subset of tasks $S \subseteq T$, we use the standard notation $d(S) := \sum_{i \in S} d(i)$ and $w(S) := \sum_{i \in S} w(i)$.

2 A p-EPTAS for BagUFP

In this section we prove Theorem 3. For the remaining of this section, fix an instance I of BagUFP and an error parameter $0 < \varepsilon < \frac{1}{2}$. Let the set of *heavy* tasks in I be

$$H = \left\{ e \in E \mid w(e) > \frac{\varepsilon \cdot \text{opt}}{m} \right\}.$$

The remaining tasks $T \setminus H$ are *light*. Our first goal is to find the set of heavy tasks in a nearly-optimal solution. Notice that a naive enumeration takes $n^{\Omega(\frac{m}{\varepsilon})}$ time, which is far from the running time of a p-EPTAS. To avoid this issue, we compute a (small enough) representative set, which is defined as follows.

► **Definition 6.** For some $R \subseteq T$, we say that R is an ε -representative set of I if there is a solution S of I such that the following holds.

1. $S \cap H \subseteq R$.
2. $w(S) \geq (1 - 3\varepsilon) \cdot \text{opt}$.

Define $q(\varepsilon, m) = \left\lceil 4m \cdot \varepsilon^{-\lceil \varepsilon^{-1} \rceil} \right\rceil$ (the meaning of $q(\varepsilon, m)$ becomes clear in Section 2.2).

► **Lemma 7.** There is an algorithm *RepSet* that, given a BagUFP instance $I = (G, u, T, P, d, w, \mathcal{B})$, $0 < \varepsilon < \frac{1}{2}$, and $\text{opt} \in [w(T)]$, in time $m^3 \cdot \varepsilon^{-2} \cdot |I|^{O(1)}$ returns $R \subseteq T$ with $|R| \leq 3 \cdot m^3 \cdot \varepsilon^{-2} \cdot q(\varepsilon, m)$. Furthermore, if $\frac{\text{opt}}{2} < \tilde{\text{opt}} \leq \text{opt}$, R is an ε -representative set of I .

In Section 2.1 we use the representative set from Lemma 7 to design a p-EPTAS for BagUFP. Then, in Section 2.2 we prove Lemma 7.

Algorithm 1 p -EPTAS(I, ε).

input : BagUFP instance I and an error parameter $0 < \varepsilon < \frac{1}{2}$.
output : A $(1 - 7\varepsilon)$ -approximate solution A for I .

- 1 $A \leftarrow \emptyset$.
- 2 **for** $\tilde{\text{opt}} \in \{1, 2, \dots, 2^{\lceil \log_2(w(T)) \rceil}\}$ **do**
- 3 Construct $R^{\tilde{\text{opt}}} \leftarrow \text{RepSet}(I, \varepsilon, \tilde{\text{opt}})$.
- 4 **for** $F \subseteq R^{\tilde{\text{opt}}}$ s.t. $|F| \leq m/\varepsilon$ and F is a feasible solution for I **do**
- 5 Compute a basic optimal solution $\lambda^{\text{opt}, F}$ for LP_F^{opt} .
- 6 Define $L_F^{\tilde{\text{opt}}} := \{i \in T_F^{\text{opt}} \mid \lambda_i^{\text{opt}, F} = 1\}$ and $A_F^{\tilde{\text{opt}}} = L_F^{\tilde{\text{opt}}} \cup F$.
- 7 **if** $w(A_F^{\tilde{\text{opt}}}) > w(A)$ **then**
- 8 $A \leftarrow A_F^{\tilde{\text{opt}}}$.
- 9 **end**
- 10 **end**
- 11 **end**
- 12 Return A .

2.1 A Representative Set Based p -EPTAS

Given the representative set algorithm described in Lemma 7, we obtain a p -EPTAS as follows (the pseudocode is given in Algorithm 1). We consider the powers of two $\tilde{\text{opt}}$ in the domain $[w(T)]$ (i.e., all values $\tilde{\text{opt}} = 1, 2, 4, \dots, 2^{\lceil \log w(T) \rceil}$). We apply the algorithm from Lemma 7 with this parameter $\tilde{\text{opt}}$ to obtain a set $R^{\tilde{\text{opt}}}$. Notice that, for $\frac{\text{opt}}{2} < \tilde{\text{opt}} \leq \text{opt}$, $R^{\tilde{\text{opt}}}$ is a representative set. Now we enumerate over all the feasible solutions $F \subseteq R^{\tilde{\text{opt}}}$ of cardinality at most m/ε . For each such F , we compute a feasible solution $A_F^{\tilde{\text{opt}}}$ (including F), and return the best such solution.

It remains to describe how $A_F^{\tilde{\text{opt}}}$ is computed. First of all, we define a reduced BagUFP instance $I_F^{\tilde{\text{opt}}} = (G, u_F, T_F^{\tilde{\text{opt}}}, P, d, w, \mathcal{B}_F)$ as follows. \mathcal{B}_F is the subset of input bags not containing any task in F . The set of tasks $T_F^{\tilde{\text{opt}}}$ is given by the tasks of weight at most $\frac{2\varepsilon}{m}\tilde{\text{opt}}$ which are contained in the bags \mathcal{B}_F . The capacity function u_F is given by $u_F(e) := u(e) - \sum_{i \in F: e \in P(i)} d(i)$ (i.e., the residual capacity after accommodating the tasks in F). Observe that, for any feasible solution L for $I_F^{\tilde{\text{opt}}}$, $L \cup F$ is a feasible solution for the input problem. Indeed, the capacity constraints are satisfied and at most one task per bag can be selected.

Given the above instance $I_F^{\tilde{\text{opt}}}$, we considering the following LP relaxation $LP_F^{\tilde{\text{opt}}}$:

$$\begin{aligned}
 \max \quad & \sum_{i \in T_F^{\tilde{\text{opt}}}} x_i \cdot w(i) && (LP_F^{\tilde{\text{opt}}}) \\
 \text{s.t.} \quad & \sum_{i \in T_F^{\tilde{\text{opt}}}: e \in P(i)} x_i \cdot d(i) \leq u_F(e) && \forall e \in E \\
 & \sum_{i \in T_F^{\tilde{\text{opt}}} \cap B} x_i \leq 1 && \forall B \in \mathcal{B}_F \\
 & x_i \geq 0 && \forall i \in T_F^{\tilde{\text{opt}}}
 \end{aligned}$$

We compute a basic optimal solution $\lambda^{\text{opt},F}$ for the above LP. Let $L_F^{\text{opt}} \subseteq T_F^{\text{opt}}$ be the tasks such that $\lambda_i^{\text{opt},F} = 1$. We set $A_F^{\text{opt}} = L_F^{\text{opt}} \cup F$. This concludes the description of the algorithm.

Obviously the above algorithm computes a feasible solution.

► **Lemma 8.** *Algorithm 1 returns a feasible solution.*

Proof. Consider a given pair (opt, F) . Obviously L_F^{opt} is a feasible solution for the BagUFP instance I_F^{opt} . Indeed, the demand of the tasks in L_F^{opt} whose path contains a given edge e is upper bounded by $\sum_{i \in T_F^{\text{opt}}: e \in P(i)} \lambda_i^{\text{opt},F} \cdot d(i) \leq u_F(e)$. Furthermore, for a given bag $B \in \mathcal{B}_F$, at most one variable $\lambda_i^{\text{opt},F}$ with $i \in B$ can be equal to 1, hence $|L_F^{\text{opt}} \cap B| \leq 1$. Thus, as argued before, $A_F^{\text{opt}} = L_F^{\text{opt}} \cup F$ is a feasible solution for the input BagUFP instance I . Since the returned solution A is one of the feasible solutions A_F^{opt} (or the empty set, which is a feasible solution), A is a feasible solution. ◀

It is also not hard to upper bound the running time.

► **Lemma 9.** *Algorithm 1 runs in time $\left(\frac{3 \cdot m^3}{\varepsilon^2} \cdot q(\varepsilon, m)\right)^{m/\varepsilon} |I|^{O(1)}$.*

Proof. Lines 3 and 5-8 can be performed in $|I|^{O(1)}$ time. Thus the overall running time is upper bounded by $|I|^{O(1)}$ multiplied by the number of possible pairs (opt, F) . There are $O(\log w(T)) = |I|^{O(1)}$ possible choices for opt . For a fixed choice of opt , one has $|R^{\text{opt}}| \leq \frac{3m^3}{\varepsilon^2} q(\varepsilon, m)$. Since F is a subset of R^{opt} of cardinality at most m/ε , the number of possible choices for F (for the considered opt) is at most $2 \left(\frac{3m^3}{\varepsilon^2} q(\varepsilon, m)\right)^{m/\varepsilon}$. The claim follows. ◀

It remains to bound the approximation factor of the algorithm. To this aim, we critically exploit the fact that each basic solution $\lambda^{\text{opt},F}$ is almost integral: more precisely, it has at most $2m$ non-integral entries. To prove that, we use a result in [39] about the sparseness of matroid polytopes with m additional linear constraints.

► **Lemma 10.** *Each solution $\lambda^{\text{opt},F}$ computed by Algorithm 1 has at most $2m$ non-integral entries.*

Proof. The proof relies on matroid theory; for more details on the subject, we refer the reader to, e.g., [45]. Consider LP_F^{opt} for any pair (opt, F) considered by the algorithm. Let $\tilde{L}P_F^{\text{opt}}$ be the LP obtained from LP_F^{opt} by dropping the m capacity constraints $\sum_{i \in T_F^{\text{opt}}: e \in P(i)} x_i \cdot d(i) \leq u_F(e)$. $\tilde{L}P_F^{\text{opt}}$ turns out to be the standard LP for a partition matroid (in particular, in an independent set at most one task per bag can be selected, where the bags induce a partition of the tasks). In [39] it is shown that every basic solution (including an optimal one) for an LP obtained by adding m linear constraints to the standard LP for any matroid (including partition ones) has at most $2m$ non-integral entries. Hence $\lambda^{\text{opt},F}$ satisfies this property. ◀

► **Lemma 11.** *The solution A returned by Algorithm 1 satisfies $w(A) \geq (1 - 7\varepsilon)\text{opt}$.*

Proof. It is sufficient to show that some solution C_F^{opt} has large enough profit. Consider the value of opt such that $\frac{\text{opt}}{2} < \tilde{\text{opt}} \leq \text{opt}$. Notice that the algorithm considers exactly one such value since $1 \leq \text{opt} \leq w(T)$. We next show how to choose a convenient $F \subseteq R^{\text{opt}}$.

5:8 Unsplittable Flow on a Short Path

Observe that for the considered choice of $\tilde{\text{opt}}$, $R^{\tilde{\text{opt}}}$ is an ε -representative set. Let S be the solution for I guaranteed by Lemma 7 and Definition 6. Recall that $w(S) \geq (1 - 3\varepsilon)\text{opt}$ and $S \cap H \subseteq R^{\tilde{\text{opt}}}$. Since each $i \in S \cap H$ has $w(i) \geq \frac{\varepsilon}{m}\text{opt}$ by definition and since obviously $w(S \cap H) \leq w(S) \leq \text{opt}$, it must be the case that $|S \cap H| \leq \frac{m}{\varepsilon}$. This implies that there is an iteration of the algorithm (for the considered $\tilde{\text{opt}}$) that has $F = S \cap H$: we will focus on that iteration.

We claim that $w(A_{S \cap H}^{\tilde{\text{opt}}}) = w(S \cap H) + w(L_F^{\tilde{\text{opt}}}) \geq (1 - 7\varepsilon)\text{opt}$. Notice that each task $i \in S \setminus H$ has weight $w(i) < \frac{\varepsilon}{m}\text{opt} < \frac{2\varepsilon}{m}\tilde{\text{opt}}$. Furthermore, by construction $i \in S \setminus H$ is contained in a bag in $\mathcal{B}_{S \cap H}$. Hence $i \in T_{S \cap H}^{\tilde{\text{opt}}}$, which implies $S \setminus H \subseteq T_{S \cap H}^{\tilde{\text{opt}}}$. The feasibility of S implies that $\sum_{i \in S \setminus H: e \in P(i)} d(i) \leq u_F(e)$ for every edge e , and $|(S \setminus H) \cap B| \leq 1$ for every $B \in \mathcal{B}_{S \cap H}$. Therefore the integral solution s which has $s_i = 1$ for $i \in S \setminus H$ and $s_i = 0$ for the remaining entries is a feasible solution for $LP_{S \cap H}^{\tilde{\text{opt}}}$. Define $lp_{S \cap H}^{\tilde{\text{opt}}} := \sum_{i \in T_{S \cap H}^{\tilde{\text{opt}}}} w(i) \cdot \lambda_i^{\tilde{\text{opt}}, S \cap H}$ as the optimal LP value for $LP_{S \cap H}^{\tilde{\text{opt}}}$. The feasibility of s implies

$$w(S \setminus H) = \sum_{i \in T_{S \cap H}^{\tilde{\text{opt}}}} w(i) \cdot s_i \leq lp_{S \cap H}^{\tilde{\text{opt}}}.$$

On the other hand,

$$w(L_{S \cap H}^{\tilde{\text{opt}}}) \geq lp_{S \cap H}^{\tilde{\text{opt}}} - 2m \cdot \frac{2\varepsilon}{m}\tilde{\text{opt}} \geq lp_{S \cap H}^{\tilde{\text{opt}}} - 4\varepsilon \cdot \text{opt}.$$

In the first inequality above we used the fact that $\lambda^{\tilde{\text{opt}}, S \cap H}$ has at most $2m$ non-integral values (by Lemma 10), and that each $i \in T_{S \cap H}^{\tilde{\text{opt}}}$ has $w(i) \leq \frac{2\varepsilon}{m}\tilde{\text{opt}}$ by construction. In the second inequality above we used the assumption that $\tilde{\text{opt}} \leq \text{opt}$. Putting everything together:

$$\begin{aligned} w(A_{S \cap H}^{\tilde{\text{opt}}}) &= w(L_{S \cap H}^{\tilde{\text{opt}}}) + w(S \cap H) \geq lp_{S \cap H}^{\tilde{\text{opt}}} - 4\varepsilon \cdot \text{opt} + w(S \cap H) \\ &\geq w(S \setminus H) - 4\varepsilon \cdot \text{opt} + w(S \cap H) = w(S) - 4\varepsilon \cdot \text{opt} \geq (1 - 7\varepsilon)\text{opt}. \end{aligned} \quad \blacktriangleleft$$

The proof of Theorem 3 follows directly from Lemma 8, Lemma 9, and Lemma 11.

2.2 Representative Set Construction

In this section, we construct a small ε -representative set for the BagUFP instance I ; this gives the proof of Lemma 7. Let $\tilde{\text{opt}} \in [w(T)]$ be a guess of the optimum value opt . Recall that in Section 2.1 we are able to find $\tilde{\text{opt}} \in (\frac{\text{opt}}{2}, \text{opt}]$ using exponential search over the domain $[w(T)]$.

We define a partition of the heavy tasks (and some tasks that are almost heavy) into *classes*, such that tasks of the same class have roughly the same weight and have the same subpath. Specifically, let $\Phi = \{P(i) \mid i \in T\}$ be the set of unique paths in the instance and define $\eta = \lceil \log_{1-\varepsilon}(\frac{\varepsilon}{2-m}) \rceil$ as a parameter describing the number of classes. For all $\varphi \in \Phi$ and $r \in [\eta]$ define the *class* of φ and r as

$$\tilde{H}(\varphi, r) = \left\{ i \in T \mid \frac{w(i)}{2 \cdot \tilde{\text{opt}}} \in \left((1-\varepsilon)^r, (1-\varepsilon)^{r-1} \right] \text{ and } P(i) = \varphi \right\}. \quad (1)$$

In simple words, a task i belongs to class $\tilde{H}(\varphi, r)$ have weight roughly $(1-\varepsilon)^r \cdot 2 \cdot \tilde{\text{opt}}$ and the subpath of i is φ . Define

$$\tilde{H} = \bigcup_{\varphi \in \Phi, r \in [\eta]} \tilde{H}(\varphi, r)$$

as the union of classes. The parameter η is carefully chosen so that the weight of every task $i \in \tilde{H}$ satisfies $w(i) \geq \frac{\varepsilon \cdot \text{opt}}{m}$, implying that i is roughly heavy. Since $\tilde{\text{opt}} \in [\frac{\text{opt}}{2}, \text{opt}]$, it follows that $H \subseteq \tilde{H}$ and that \tilde{H} does not contain tasks with significantly smaller weight than $\frac{\varepsilon \cdot \text{opt}}{m}$ - that is the minimum weight allowed for heavy tasks.

► **Observation 12.** $H \subseteq \tilde{H}$.

Let $\mathcal{D} = \{\tilde{H}(\phi, r) \mid \phi \in \Phi, r \in [\eta]\}$ be the set of classes. We use a simple upper bound on the number of classes.

► **Lemma 13.** $|\mathcal{D}| \leq 3 \cdot m^3 \cdot \varepsilon^{-2}$.

Proof. Observe that

$$\log_{1-\varepsilon} \left(\frac{\varepsilon}{2 \cdot m} \right) \leq \frac{\ln \left(\frac{2 \cdot m}{\varepsilon} \right)}{-\ln(1-\varepsilon)} \leq \frac{2 \cdot m \cdot \varepsilon^{-1}}{\varepsilon} = 2 \cdot m \cdot \varepsilon^{-2}. \quad (2)$$

The second inequality follows from $x < -\ln(1-x), \forall x > -1, x \neq 0$, and $\ln(y) < y, \forall y > 0$. Moreover, the number of subpaths $\varphi \in \Phi$ is bounded by $|\Phi| = \binom{m}{2} \leq m^2$. Therefore, the number of classes is bounded by

$$|\mathcal{D}| \leq m^2 \cdot \left(\log_{1-\varepsilon} \left(\frac{\varepsilon}{2 \cdot m} \right) + 1 \right) \leq 2 \cdot m \cdot \varepsilon^{-2} \cdot m^2 + m^2 = 2 \cdot m^3 \cdot \varepsilon^{-2} + m^2 \leq 3 \cdot m^3 \cdot \varepsilon^{-2} \quad (3)$$

The first inequality follows from (2). ◀

■ **Algorithm 2** RepSet($I, \varepsilon, \tilde{\text{opt}}$).

input : BagUFPInstance I , an error parameter $0 < \varepsilon < \frac{1}{2}$, and $\tilde{\text{opt}} \in [w(T)]$.
output : An ε -representative set R for I (if $\tilde{\text{opt}} \in [\frac{\text{opt}}{2}, \text{opt}]$).

- 1 Initialize $R \leftarrow \emptyset$.
- 2 **forall** $\varphi \in \Phi$ and $r \in [\eta]$ **do**
- 3 Let $\mathcal{B}(\varphi, r) = \{B \in \mathcal{B} \mid B \cap \tilde{H}(\varphi, r) \neq \emptyset\}$.
- 4 For every $B \in \mathcal{B}(\varphi, r)$ define $i_B(\varphi, r) = \arg \min_{i \in B \cap \tilde{H}(\varphi, r)} d(i)$.
- 5 Sort $\mathcal{B}(\varphi, r)$ in non-decreasing order $B_1(\varphi, r), \dots, B_\ell(\varphi, r)$
 by $d(i_B(\varphi, r)) \quad \forall B \in \mathcal{B}(\varphi, r)$.
- 6 Define $a = \min \{q(\varepsilon, m), |\mathcal{B}(\varphi, r)|\}$.
- 7 Update $R \leftarrow R \cup \{i_{B_1}(\varphi, r), \dots, i_{B_a}(\varphi, r)\}$.
- 8 **end**
- 9 Return R .

Our representative set construction is fairly simple. For each class $\tilde{H}(\varphi, r)$, consider the set of *active* bags $\mathcal{B}(\varphi, r)$ for $\tilde{H}(\varphi, r)$ that contain at least one task in $\tilde{H}(\varphi, r)$. For every active bag $B \in \mathcal{B}(\varphi, r)$ define the *representative* of B in the class $\tilde{H}(\varphi, r)$ as the task from the bag B in the class $\tilde{H}(\varphi, r)$ of minimum demand (if there is more than one such task we choose one arbitrarily). We sort the active bags of the class in a non-decreasing order according to the demand of the representatives of the bags. Finally, we take the first a representatives (at most one from each bag) according to this order, where a is the minimum between the parameter $q(\varepsilon, m)$ and the number of active bags for the class. The pseudocode of the algorithm is given in Algorithm 2.

We give an outline of the proof of Lemma 7. Consider some optimal solution OPT for the instance. We partition the tasks in OPT into three sets: L, J_{k^*} , and Q such that (i) the maximum weight of a task in Q is at most ε -times the minimum weight of a task in L ; (ii) L

5:10 Unsplittable Flow on a Short Path

is small: $|L| \leq \frac{q(\varepsilon, m)}{2}$; (iii) The weight of J_{k^*} is small: $w(J_{k^*}) \leq \varepsilon \cdot \text{opt}$. To prove that R is a representative set, we need to replace $H \cap \text{OPT}$ with tasks from R . As a first step, we define a mapping h from $\tilde{H} \cap \text{OPT}$ to R , where each task $i \in \tilde{H} \cap \text{OPT}$ is replaced by a task from the same class of a smaller or equal demand. For tasks $i \in \tilde{H} \cap \text{OPT}$ such that R contains a representative from the bag of i in the class of i , we simply define $h(i)$ as this representative; for other tasks, we define the mapping via a bipartite matching on the remaining tasks and representatives.

We define a solution S satisfying the conditions of Definition 6 in two steps. First, we define initial solutions S_1, S_2 . The solution S_1 contains the mapping $h(i)$ of every $i \in \tilde{H} \cap \text{OPT}$ and the tasks in $L \setminus \tilde{H}$; the solution S_2 contains all tasks in Q from bags that do not contain tasks from S_1 . Finally, we define $S = S_1 \cup S_2$. By the properties of L, J_{k^*} , and Q we are able to show that S is roughly an optimal solution. Specifically, by (iii) discarding J_{k^*} from the solution S does not have a significant effect on the total weight of S . Additionally, by property (i) there is a large gap between the weights in S_1 and S_2 ; thus, combined with property (ii) we lose only a small factor due to tasks discarded from Q , and it follows that the weight of S is $(1 - O(\varepsilon)) \cdot \text{opt}$.

Proof of Lemma 7

We start with the running time analysis of the algorithm.

▷ **Claim 14.** The running time of Algorithm 2 is bounded by $m^3 \cdot \varepsilon^{-2} \cdot |I|^{O(1)}$ on input I, ε , and opt . Moreover, $|R| \leq 3 \cdot m^3 \cdot \varepsilon^{-2} \cdot q(\varepsilon, m)$.

Proof. Each iteration of the **for** loop of the algorithm can be trivially computed in time $|I|^{O(1)}$. In addition, the number of iterations of the **for** loop is bounded by $3 \cdot m^3 \cdot \varepsilon^{-2}$ using Lemma 13. Therefore, the running time of the algorithm is bounded by $m^3 \cdot \varepsilon^{-2} \cdot |I|^{O(1)}$. For the second property of the lemma, recall that the number of classes is bounded by $3 \cdot m^3 \cdot \varepsilon^{-2}$ using Lemma 13. By Algorithm 2 of the algorithm, the number of tasks taken to R from each class is at most $q(\varepsilon, m)$. Therefore, $|R| \leq 3 \cdot m^3 \cdot \varepsilon^{-2} \cdot q(\varepsilon, m)$. ◁

If $\text{opt} \notin [\frac{\text{opt}}{2}, \text{opt}]$, the proof immediately follows from Claim 14. Thus, for the following assume that $\text{opt} \in [\frac{\text{opt}}{2}, \text{opt}]$. Let $\text{OPT} \subseteq T$ be an optimal solution for I . Let $w^* = \frac{\varepsilon \cdot \text{opt}}{m}$ be a lower bound on the minimum weight of a task in \tilde{H} . We partition a subset of the tasks in $\text{OPT} \setminus \tilde{H}$ with the highest weights into $N = \lceil \varepsilon^{-1} \rceil$ disjoint sets. For all $k \in [N]$ define the k -th set as

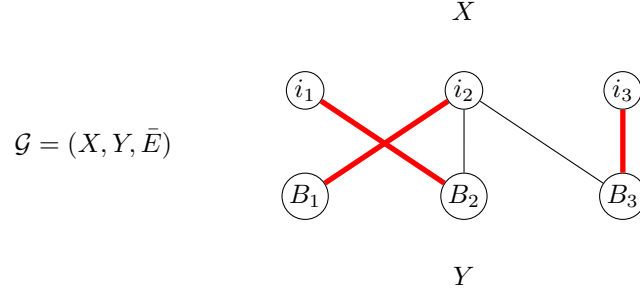
$$J_k = \{i \in \text{OPT} \setminus \tilde{H} \mid w(i) \in (\varepsilon^k \cdot w^*, \varepsilon^{k-1} \cdot w^*)\}. \quad (4)$$

Let $k^* = \arg \min_{k \in [N]} w(J_k)$. By (4) the sets J_1, \dots, J_N are $N \geq \varepsilon^{-1}$ disjoint sets (some of them may be empty); thus, $w(J_{k^*}) \leq \varepsilon \cdot \text{opt}$. Define

$$L = (\text{OPT} \cap \tilde{H}) \cup \bigcup_{k \in [k^* - 1]} J_k$$

as the subset of all tasks in OPT of weight greater than $\varepsilon^{k^* - 1} \cdot w^*$, and define $Q = \text{OPT} \setminus (L \cup J_{k^*})$ as the remaining tasks in OPT excluding J_{k^*} . We use the following auxiliary claim.

▷ **Claim 15.** $|L| \leq \frac{q(\varepsilon, m)}{2}$.



■ **Figure 1** An illustration of the graph \mathcal{G} and the maximum matching M (in red). Every edge (i, B) in the graph indicates that bag B belongs to $\text{fit}(i)$; that is, the representative from B in the class of i belongs to R and the demand of this representative is at most the demand of i . Note that even though i_1 and i_2 are both connected to bag B_2 , i_1 and i_2 may belong to different classes.

Proof. If $L = \emptyset$ the claim trivially follows. Otherwise,

$$|L| \leq \sum_{i \in L} \frac{w(i)}{\varepsilon^{k^*-1} \cdot w^*} = \frac{w(L)}{\varepsilon^{k^*-1} \cdot w^*} \leq \frac{\text{opt}}{\varepsilon^{k^*-1} \cdot w^*} \quad (5)$$

The first inequality holds since $w(i) \geq \varepsilon^{k^*-1} \cdot w^*$ for all $i \in L$. The second inequality follows from the fact that $L \subseteq \text{OPT}$; thus, L is a solution for I . Thus, by (5) and the definition of w^*

$$|L| \leq \frac{\text{opt}}{\varepsilon^{k^*-1} \cdot 2\tilde{\text{opt}} \cdot \frac{\varepsilon}{2 \cdot m}} \leq \frac{\text{opt}}{\varepsilon^{k^*-1} \cdot \text{opt} \cdot \frac{\varepsilon}{2 \cdot m}} = \frac{2 \cdot m}{\varepsilon^{k^*}} \leq \frac{2 \cdot m}{\varepsilon^N} \leq \frac{q(\varepsilon, m)}{2}.$$

The second inequality holds since we assume that $\tilde{\text{opt}} \geq \frac{\text{opt}}{2}$. \triangleleft

Let R be the set returned by the algorithm. In the following, we show the existence of a solution S such that $S \cap H \subseteq R$ and $w(S) \geq (1 - 3 \cdot \varepsilon) \cdot \text{opt}$; this gives the statement of the lemma by Definition 6. To construct S , we first define a mapping h from $\tilde{H} \cap \text{OPT}$ to R . For a subpath $\varphi \in \Phi$ and $r \in [\eta]$, recall the set of active bags $\mathcal{B}(\varphi, r)$ and the representatives $i_B(\varphi, r)$ for all $B \in \mathcal{B}(\varphi, r)$ (see Algorithm 2).

For the simplicity of the notation, for $\varphi \in \Phi$, $r \in [\eta]$, and $i \in \tilde{H}(\varphi, r)$ let $\tilde{H}^i = \tilde{H}(\varphi, r)$ be the class to which i belongs and let $r_i = r$; moreover, for $B \in \mathcal{B}$ such that $i \in B$ define $B^i = B$ as the bag containing i . We first consider tasks i in $\text{OPT} \cap \tilde{H}$ whose bag does not have a representative in R from the class of i , i.e., $R \cap \tilde{H}^i \cap B^i = \emptyset$. Define this set of tasks as

$$X = \left\{ i \in \text{OPT} \cap \tilde{H} \mid R \cap \tilde{H}(\varphi, r) \cap B^i = \emptyset \right\}. \quad (6)$$

The above set X contains all tasks $i \in \text{OPT} \cap \tilde{H}$ whose corresponding bag does not have a representative in R from the class of i . We define a bipartite graph, in which X is one side of the graph. The other side of the graph is

$$Y = \mathcal{B} \setminus \{B \in \mathcal{B} \mid \exists i \in L \text{ s.t. } B = B^i\}. \quad (7)$$

In words, Y describes all *available* bags, the collection of all bags that do not contain a task in L . Define the bipartite graph $\mathcal{G} = (X, Y, \bar{E})$ such that the set of edges is defined as follows. For some $i \in X$ let

$$\text{fit}(i) = \left\{ B \in Y \mid B \cap R \cap \tilde{H}^i \neq \emptyset \text{ and } d(i_B(P(i), r_i)) \leq d(i) \right\}. \quad (8)$$

5:12 Unsplittable Flow on a Short Path

The set $\text{fit}(i)$ describes all bags that can potentially be matched to i ; these bags have a representative from the class $\tilde{H}^i = \tilde{H}(P(i), r_i)$ that contain i and the representative of the bag has a smaller or equal demand w.r.t. i . Now, a task i can be matched to a bag B only if $B \in \text{fit}(i)$, i.e., define

$$\bar{E} = \left\{ (i, B) \in X \times Y \mid B \in \text{fit}(i) \right\}. \quad (9)$$

Let M be a maximum matching in \mathcal{G} . We give an illustration of the above construction in Figure 1. We show that M matches all vertices in X .

▷ **Claim 16.** For every $i \in X$ there is $B \in Y$ such that $(i, B) \in M$.

Proof. Assume towards a contradiction that there is $i \in X$ such that for all $B \in Y$ it holds that $(i, B) \notin M$. Let $\varphi \in \Phi$ and $r \in [\eta]$ such that $\tilde{H}^i = \tilde{H}(\varphi, r)$. Since $i \in X$, by (6) it holds that $R \cap \tilde{H}(\varphi, r) \cap B^i = \emptyset$. Intuitively, this means that the algorithm preferred other bags over B^i in the selection of representatives for class $\tilde{H}(\varphi, r)$. Therefore, by Algorithm 2 of the algorithm, there are $q(\varepsilon, m)$ distinct bags $B_1 = B_1(\varphi, r), \dots, B_{q(\varepsilon, m)} = B_{q(\varepsilon, m)}(\varphi, r)$ such that for all $j \in [q(\varepsilon, m)]$ it holds that $i_{B_j}(\varphi, r) \in R$ and $d(i_{B_j}(\varphi, r)) \leq d(i)$. Thus, for all $j \in [q(\varepsilon, m)]$ it holds that $(i, B_j) \in \bar{E}$ by (8) and (9). In addition,

$$|M| \leq |X| \leq |L| \leq \frac{q(\varepsilon, m)}{2} < q(\varepsilon, m). \quad (10)$$

The first inequality holds since M is a matching in \mathcal{G} and X is one side of a bipartition of \mathcal{G} . The second inequality holds since $X \subseteq L$ by (6) and the definition of L . The third inequality follows from Claim 15. The last inequality holds since $q(\varepsilon, m) \geq 2$ assuming $0 < \varepsilon < \frac{1}{2}$ and $m \geq 1$. By (10) there is $j \in [q(\varepsilon, m)]$ such that for all $t \in X$ it holds that $(t, B_j) \notin M$. In particular, $(i, B_j) \notin M$ and recall that $(i, B_j) \in \bar{E}$. Therefore, $M \cup (i, B_j)$ is a matching in \mathcal{G} in contradiction that M is a maximum matching in \mathcal{G} . ◁

For every $i \in X$ define $M_i = B$ such that $(i, B) \in M$, i.e., M_i is the bag matched to i in M . By Claim 16 it holds that each task in X is matched and every bag is matched at most once. We define the mapping h from $\tilde{H} \cap \text{OPT}$ to R . Define $h : \tilde{H} \cap \text{OPT} \rightarrow R$ such that for all $i \in \tilde{H} \cap \text{OPT}$:

$$h(i) = \begin{cases} i_{B^i}(P(i), r_i), & \text{if } B^i \cap R \cap \tilde{H}^i \neq \emptyset \\ i_{M_i}(P(i), r_i), & \text{else} \end{cases} \quad (11)$$

In words, a task $i \in \tilde{H} \cap \text{OPT}$ is mapped to a task $h(i)$ such that if the bag of i contains a representative in R in the class of i - then $h(i)$ is this representative; otherwise, $h(i)$ is the representative of the bag M_i matched to i by the matching M . Clearly, h is well defined by Claim 16. We list immediate properties of h .

► **Observation 17.** The function h satisfies the following.

- For every $i \in \tilde{H} \cap \text{OPT}$ it holds that $d(h(i)) \leq d(i)$ and $\tilde{H}^{h(i)} = \tilde{H}^i$.
- For every $i, j \in \tilde{H} \cap \text{OPT}$, $i \neq j$, it holds that $B^{h(i)} \neq B^{h(j)}$.
- For every $i \in \tilde{H} \cap \text{OPT}$ and $t \in L \setminus \tilde{H}$ it holds that $B^{h(i)} \neq B^t$.

The first property follows from the definition of the graph \mathcal{G} and the definition of the bag representatives in Algorithm 2. The second and third properties hold since OPT takes at most one task from each bag and using the definition of \mathcal{G} . We can finally define the solution S that satisfies the conditions of Definition 6. Define

$$S_1 = \{h(i) \mid i \in \tilde{H} \cap \text{OPT}\} \cup (L \setminus \tilde{H}) \quad (12)$$

and

$$S_2 = \{i \in Q \mid B^i \neq B^t \ \forall t \in S_1\}. \quad (13)$$

Define $S = S_1 \cup S_2$. We show that S satisfies the conditions of Definition 6. As an immediate property of the construction we have the following.

► **Observation 18.** h is a one-to-one function from $\tilde{H} \cap \text{OPT}$ to $S \cap \tilde{H}$.

We use the above to prove the feasibility of S .

▷ **Claim 19.** S is a solution for I .

Proof. We show that S satisfies the bag constraints. Let $B \in \mathcal{B}$. Since OPT is a solution for I , there is at most one $i \in B \cap \text{OPT}$. We consider four cases depending on the task i .

1. If $i \in \tilde{H}$ and $R \cap \tilde{H}^i \cap B^i \neq \emptyset$. Then, $h(i) \in B$ by (11) and for all $t \in S_1 \setminus \{h(i)\}$ it holds that $t \notin B$ by Observation 17. Furthermore, for all $t \in S_2$ it holds that $t \notin B$ by (13). Thus, $|B \cap S| \leq 1$.
2. If $i \in \tilde{H}$ and $R \cap \tilde{H}^i \cap B^i = \emptyset$. Then, as $\tilde{H} \subseteq L$ it holds that $i \in L$; thus, $B \notin Y$ by (7). Therefore, by (12) we conclude that $|B \cap S_1| = 0$; thus,

$$|B \cap S| = |B \cap S_2| \leq |B \cap Q| \leq |B \cap \text{OPT}| \leq 1.$$

The equality holds since $|B \cap S_1| = 0$. The first inequality follows from (13). The last inequality holds since OPT is a solution.

3. If $i \in L \setminus \tilde{H}$. Then, by (12) and (13) it holds that $|B \cap S| = |B \cap i| = 1$.
4. If $i \in Q$. Then, there are two sub cases. If $i \in S_2$, by (13) for all $t \in S_1$ it holds that $B \neq B^t$; thus, as $|Q \cap B| \leq |\text{OPT} \cap B| \leq 1$ it follows that $|S \cap B| \leq 1$. Otherwise, $i \notin S_2$; then, by (13) it holds that

$$|B \cap S| = |B \cap S_1| \leq 1.$$

The inequality follows from Observation 17.

By the above we conclude that S satisfies all bag constraints. It remains to prove that S satisfies the capacity constraints of all edges. For $e \in E$

$$\begin{aligned} \sum_{i \in S \text{ s.t. } e \in P(i)} d(i) &= \sum_{i \in S \cap \tilde{H} \text{ s.t. } e \in P(i)} d(i) + \sum_{i \in S \setminus \tilde{H} \text{ s.t. } e \in P(i)} d(i) \\ &= \sum_{i \in \text{OPT} \cap \tilde{H} \text{ s.t. } e \in P(i)} d(h(i)) + \sum_{i \in S \setminus \tilde{H} \text{ s.t. } e \in P(i)} d(i) \\ &\leq \sum_{i \in \text{OPT} \cap \tilde{H} \text{ s.t. } e \in P(i)} d(h(i)) + \sum_{i \in \text{OPT} \setminus \tilde{H} \text{ s.t. } e \in P(i)} d(i) \\ &\leq \sum_{i \in \text{OPT} \cap \tilde{H} \text{ s.t. } e \in P(i)} d(i) + \sum_{i \in \text{OPT} \setminus \tilde{H} \text{ s.t. } e \in P(i)} d(i) \\ &= \sum_{i \in \text{OPT} \text{ s.t. } e \in P(i)} d(i) \\ &\leq u(e). \end{aligned}$$

The second equality holds since h is a one-to-one mapping from $\text{OPT} \cap \tilde{H}$ to $S \cap \tilde{H}$ by Observation 18. The first inequality holds since $S \setminus \tilde{H} \subseteq \text{OPT} \setminus \tilde{H}$ by (12) and (13). The second inequality holds since $d(h(i)) \leq d(i)$ for all $i \in \text{OPT} \cap \tilde{H}$ by Observation 17. The last inequality holds since OPT is a solution for I . ◁

5:14 Unsplittable Flow on a Short Path

Observe that there is a substantial gap in weight between tasks in L and tasks in Q . We use this gap in the following auxiliary claim.

▷ **Claim 20.** $w(Q \setminus S) \leq \varepsilon \cdot \text{opt}$.

Proof. Observe that

$$|Q \setminus S| = |Q \setminus S_2| = |\{i \in Q \mid \exists t \in S_1 \text{ s.t. } B^i = B^t\}| \leq |S_1| = |L|. \quad (14)$$

The inequality holds since Q satisfies the bag constraints (i.e., $|Q \cap B| \leq 1$ for all $B \in \mathcal{B}$); thus, for each $t \in S_1$ there can be at most one $i \in Q$ such that $B^i = B^t$ (and only in this case i is discarded from S_2). The last equality holds since h is a one-to-one mapping from $\text{OPT} \cap \tilde{H}$ to $S \cap \tilde{H}$ by Observation 18 and since $L \setminus \tilde{H}$ belongs both to S_1 and L . Hence,

$$w(Q \setminus S) \leq |Q \setminus S| \cdot \varepsilon^{k^*} \cdot w^* \leq |L| \cdot \varepsilon^{k^*} \cdot w^* \leq \varepsilon \cdot w(L) \leq \varepsilon \cdot w(\text{OPT}) = \varepsilon \cdot \text{opt}.$$

The first inequality holds since $w(i) \leq \varepsilon^{k^*} \cdot w^*$ for all $i \in Q$. The second inequality follows from (14). The third inequality holds since $w(i) > \varepsilon^{k^*-1} \cdot w^*$ for all $i \in L$. The last inequality holds since $L \subseteq \text{OPT}$. ◁

The following claim shows that S satisfies the total weight required by Definition 6.

▷ **Claim 21.** $w(S) \geq (1 - 3\varepsilon) \cdot \text{opt}$.

Proof. We first give a lower bound to the weight of S_1 .

$$\begin{aligned} w(S_1) &= w((L \setminus \tilde{H}) \cup \{h(i) \mid i \in \tilde{H} \cap \text{OPT}\}) \\ &= w(L \setminus \tilde{H}) + \sum_{i \in \tilde{H} \cap \text{OPT}} w(h(i)) \\ &\geq w(L \setminus \tilde{H}) + \sum_{i \in \tilde{H} \cap \text{OPT}} (1 - \varepsilon) \cdot w(i) \\ &\geq (1 - \varepsilon) \cdot w(L). \end{aligned} \quad (15)$$

The inequality holds since for all $i \in \text{OPT} \cap \tilde{H}$ it holds that $\tilde{H}^i = W^{h(i)}$ by Observation 17; thus, by (1) it follows that $w(h(i)) \geq (1 - \varepsilon) \cdot w(i)$. For the last inequality, recall that $\tilde{H} \subseteq L$. Moreover,

$$w(S_2) = w(Q) - w(Q \setminus S) \geq w(Q) - \varepsilon \cdot \text{opt} \geq (1 - \varepsilon) \cdot w(Q) - \varepsilon \cdot \text{opt}. \quad (16)$$

The first equality holds since $S_2 \subseteq Q$. The first inequality follows from Claim 20. By (15) and (16) we have

$$\begin{aligned} w(S) &= w(S_1) + w(S_2) \\ &\geq (1 - \varepsilon) \cdot w(L \cup Q) - \varepsilon \cdot \text{opt} \\ &= (1 - \varepsilon) \cdot w(\text{OPT} \setminus J_{k^*}) - \varepsilon \cdot \text{opt} \\ &\geq (1 - \varepsilon) \cdot (1 - \varepsilon) \cdot \text{opt} - \varepsilon \cdot \text{opt} \\ &\geq (1 - 3\varepsilon) \cdot \text{opt}. \end{aligned}$$

The second inequality holds since $w(J_{k^*}) \leq \varepsilon \cdot \text{opt}$. ◁

Observe that $H \subseteq \tilde{H}$ by Observation 12. Moreover, $S \cap \tilde{H} = S_1 \cap \tilde{H} \subseteq R$ by (12) and (13). Thus, $S \cap H \subseteq R$. By Claim 19 and Claim 21, it follows that R is a representative set.

The proof follows from Claim 19, Claim 21, and Claim 14. ◀

3 A Faster p-EPTAS for UFP

In this section we prove Theorem 5. Let (G, u, T, P, d, w) be a UFP instance. For simplicity, we next assume that $1/\varepsilon$ is integer and that $n \gg 1/\varepsilon$. Recall that $\Phi = \{P(i) \mid i \in T\}$ is the set of unique paths in the instance, and for every $\varphi \in \Phi$ we use $T_\varphi = \{i \in T \mid P(i) = \varphi\}$ to denote the set of tasks with path φ . Observe that $|\Phi| \leq \frac{1}{2} \cdot m \cdot (m+1)$.

See Algorithm 3 for a pseudocode description of our approach.

Algorithm 3 p-EPTAS for UFP.

input : UFP instance $I = (G, u, T, P, d, w)$ and a parameter $0 < \varepsilon < 0.1$
output : A feasible solution APX for the instance I
Notations: Here $w_{\max} = \max_{i \in T} w(i)$ and $p_\varphi := \sum_{i \in T_\varphi} p(i)$ for all $\varphi \in \Phi$.

- 1 Define $p(i) \leftarrow \left\lfloor \frac{n \cdot w(i)}{\varepsilon \cdot w_{\max}} \right\rfloor$ for every $i \in T$.
- 2 For all $\varphi \in \Phi$ compute DP_φ for the Knapsack instance $(T_\varphi, d, p, \min_{e \in \varphi} u(e))$.
- 3 $\text{APX} \leftarrow \emptyset$.
- 4 **for** all the powers $\tilde{\text{opt}}$ of $(1 + \varepsilon)$ in $\left[1, \frac{n^2}{\varepsilon}\right]$ **do**
- 5 **for** all non-negative integers $(X_\varphi)_{\varphi \in \Phi}$ such that $\sum_{\varphi \in \Phi} X_\varphi \leq |\Phi| \cdot \frac{1+\varepsilon}{\varepsilon}$ **do**
- 6 Set $\text{opt}_\varphi = X_\varphi \cdot \frac{\varepsilon}{|\Phi|} \cdot \tilde{\text{opt}}$ for all $\varphi \in \Phi$.
- 7 $\text{APX}' \leftarrow \bigcup_{\varphi \in \Phi} \text{DP}_\varphi(\min\{p_\varphi, \lceil \text{opt}_\varphi \rceil\})$.
- 8 **if** APX' is a feasible solution for I and $p(\text{APX}') \geq p(\text{APX})$ **then**
- 9 $\text{APX} \leftarrow \text{APX}'$.
- 5 **end**
- 4 **end**
- 10 **end**
- 11 Return APX.

We start to perform a standard rounding of the weights (similar to several other packing problems) so that they are positive integers in a polynomially bounded range. Let $w_{\max} = \max_{i \in T} w_i$ be the maximum weight of any task. Observe that, since w.l.o.g. each task alone induces a feasible solution, one has $\text{opt} \geq w_{\max}$. We replace each weight $w(i)$ with $p(i) := \left\lfloor \frac{n \cdot w(i)}{\varepsilon \cdot w_{\max}} \right\rfloor$. A standard calculation shows that an optimum solution OPT' computed w.r.t. the modified weights p is a $(1 - \varepsilon)$ -approximate solution w.r.t. the original problem. Now the (rounded) weights are in the range $\left[\frac{n}{\varepsilon}\right]$. With the obvious notation, for $S \subseteq T$, we will denote $p(S) := \sum_{i \in S} p(i)$.

Now we proceed by describing the two main phases of our p-EPTAS. In the first phase we consider each path $\varphi \in \Phi$, and define a Knapsack instance $K_\varphi = (T_\varphi, d, p, \min_{e \in \varphi} u(e))$. Here T_φ is the set of items that can be placed in the knapsack, $d(i)$ and $p(i)$ are the size and profit of item $i \in T_\varphi$, resp., and $\min_{e \in \varphi} u(e)$ is the size of the knapsack. We solve this instance K_φ using the standard algorithm for Knapsack based on dynamic programming. In more detail, this algorithm defines a dynamic programming table DP_φ indexed by the possible values $p' \in [p_\varphi]$ of the profit, where $p_\varphi := \sum_{i \in T_\varphi} p(i)$. At the end of the algorithm, for each such p' , $\text{DP}_\varphi(p')$ contains a subset of items (in T_φ) of minimum total size (or, equivalently, demand) whose profit is at least p'^2 . Notice that T_φ satisfies $p(T_\varphi) = p_\varphi \geq p'$, hence all the

² In a more standard version of the algorithm $\text{DP}_\varphi(p')$ would contain a minimum size solution of profit *exactly* p' , or a special character if such solution does not exist. However, it is easy to adapt the algorithm to rather obtain the desired values.

5:16 Unsplittable Flow on a Short Path

table entries $\text{DP}_\varphi(p')$ are well defined. We also remark that certain table entries may contain a solution of total demand larger than $\min_{e \in \varphi} u(e)$, hence such entries will never be used to construct a feasible UFP solution. Computing the dynamic tables for all $\varphi \in \Phi$ takes time

$$\sum_{\varphi \in \Phi} O(|T_\varphi| \cdot p_\varphi) \leq \sum_{\varphi \in \Phi} O(|T_\varphi|) \cdot \sum_{\varphi \in \Phi} O(p_\varphi) = O(|T|) \cdot O(p(T)) \leq O\left(\frac{n^3}{\varepsilon}\right).$$

We store these dynamic tables for later use.

At this point the second phase of the algorithm starts. Let $\text{opt}' = p(\text{OPT}')$, where OPT' is an optimal solution for the UFP instance with the rounded weights p (i.e., (G, u, T, P, d, p)). Observe that $\text{opt}' \in \left[\frac{n^2}{\varepsilon}\right]$. Let $\tilde{\text{opt}}$ be a power of $(1 + \varepsilon)$ such that $\frac{\text{opt}'}{1+\varepsilon} < \tilde{\text{opt}} \leq \text{opt}'$.

We can find this value by trying all the $O\left(\log_{1+\varepsilon} \frac{n^2}{\varepsilon}\right) = O\left(\frac{1}{\varepsilon} \cdot \log n\right)$ possibilities. Define $\text{OPT}'_\varphi := \text{OPT}' \cap T_\varphi$ and $\text{opt}'_\varphi = p(\text{OPT}'_\varphi)$. For each $\varphi \in \Phi$ we guess the largest multiple $\tilde{\text{opt}}_\varphi = X_\varphi \cdot \frac{\varepsilon}{|\Phi|} \cdot \tilde{\text{opt}}$ of $\frac{\varepsilon}{|\Phi|} \cdot \tilde{\text{opt}}$ which is upper bounded by opt'_φ . Again by guessing we mean trying all the possible combinations. Obviously a valid guess must satisfy

$$\frac{\varepsilon}{|\Phi|} \cdot \tilde{\text{opt}} \cdot \sum_{\varphi \in \Phi} X_\varphi = \sum_{\varphi \in \Phi} \tilde{\text{opt}}_\varphi \leq \sum_{\varphi \in \Phi} \text{opt}'_\varphi = \text{opt}' \leq (1 + \varepsilon)\tilde{\text{opt}},$$

hence $\sum_{\varphi \in \Phi} X_\varphi \leq Y := \lfloor \frac{1+\varepsilon}{\varepsilon} |\Phi| \rfloor$. Thus it is sufficient to generate all the ordered sequences of $|\Phi|$ non-negative integers whose sum is at most Y . As we will argue, the number of such sequences is sufficiently small.

Given a guess $\{\tilde{\text{opt}}_\varphi\}_{\varphi \in \Phi}$, we compute a tentative solution $\text{APX}' := \cup_{\varphi \in \Phi} \text{DP}_\varphi(\min\{p_\varphi, \lceil \tilde{\text{opt}}_\varphi \rceil\})$ using the pre-computed dynamic tables. Notice that, for a valid guess of $\tilde{\text{opt}}_\varphi$, by integrality we also have $\lceil \tilde{\text{opt}}_\varphi \rceil \leq \text{opt}'_\varphi$. Upper bounding with $p_\varphi \geq \text{opt}'_\varphi$ guarantees that the algorithm only uses well-defined table entries. Among the solutions APX' which are feasible, we return one APX of maximum profit $p(\text{APX})$. This concludes the description of the algorithm.

We can further improve the running time as follows. Let us compute and store the values $d(\text{DP}_\varphi(p'))$ and $p(\text{DP}_\varphi(p'))$ (this does not affect the asymptotic running time). In the for loops we only update the current value of $\text{apx} := p(\text{APX})$ instead of updating APX explicitly each time. Furthermore for each tentative solution APX' we only compute $p(\text{APX}')$ and $\sum_{i \in \text{APX}': e \in P(i)} d(i)$ for each $e \in E$. This can be done in $O(|\Phi|m)$ time and it is sufficient to check whether APX' is a feasible solution and whether $p(\text{APX}') > \text{apx}$. We maintain the combination of the parameters X_φ^* and $\tilde{\text{opt}}^*$ that lead to the current value of apx . At the end of the process from the optimal parameters X_φ^* and $\tilde{\text{opt}}^*$ we derive a corresponding solution APX of profit apx in $O(n + |\Phi|) = O(n^2)$ extra time.

► **Lemma 22.** *Algorithm 3 produces a feasible UFP solution.*

Proof. Obviously since $\text{APX} = \emptyset$ is a feasible solution, and whenever we update APX , we do that with the value APX' of a feasible solution. ◀

► **Lemma 23.** *Algorithm 3 produces a $(1 - 2\varepsilon)$ -approximate solution.*

Proof. Let us show that $p(\text{APX}) \geq (1 - \varepsilon)p(\text{OPT}')$. Notice that $\text{opt}' = p(\text{OPT}') \in \left[\frac{n^2}{\varepsilon}\right]$, hence there is a value $\tilde{\text{opt}}$ considered by the algorithm such that $\frac{1}{1+\varepsilon}\text{opt}' < \tilde{\text{opt}} \leq \text{opt}'$. Let us focus on execution of the external for loop with that value of $\tilde{\text{opt}}$.

Recall that $\text{opt}'_\varphi = p(\text{OPT}'_\varphi) = p(\text{OPT}' \cap T_\varphi)$. As already argued before, there are corresponding values $(X_\varphi)_{\varphi \in \Phi}$ considered by the algorithm such that $\tilde{\text{opt}}_\varphi = X_\varphi \cdot \frac{\varepsilon}{|\Phi|} \cdot \tilde{\text{opt}}$ satisfies:

$$\text{opt}'_\varphi - \frac{\varepsilon}{|\Phi|} \tilde{\text{opt}} \leq \tilde{\text{opt}}_\varphi \leq \text{opt}'_\varphi.$$

Let us focus on the execution of the inner for loop with these values of X_φ (hence $\tilde{\text{opt}}_\varphi$). The profit of the corresponding solution APX' is at least

$$\sum_{\varphi \in \Phi} \tilde{\text{opt}}_\varphi \geq \sum_{\varphi \in \Phi} \text{opt}'_\varphi - \varepsilon \tilde{\text{opt}} = \text{opt}' - \varepsilon \tilde{\text{opt}} \geq (1 - \varepsilon) \text{opt}'.$$

Observe that $\text{OPT}'_\varphi = \text{OPT}' \cap T_\varphi$ is a valid solution for the Knapsack instance K_φ with profit opt'_φ , where $p_\varphi \geq \text{opt}'_\varphi \geq \lceil \tilde{\text{opt}}_\varphi \rceil$, hence also a valid candidate solution for $\text{DP}_\varphi(\min\{p_\varphi, \lceil \tilde{\text{opt}}_\varphi \rceil\})$. As a consequence $d(\text{DP}_\varphi(\min\{p_\varphi, \lceil \tilde{\text{opt}}_\varphi \rceil\})) \leq d(\text{OPT}'_\varphi)$. We conclude that APX' is a feasible solution. In more detail, for each $e \in E$,

$$\sum_{i \in \text{APX}': e \in P(i)} d(i) = \sum_{\varphi \in \Phi: e \in \varphi} d(\text{DP}_\varphi(\lceil \tilde{\text{opt}}_\varphi \rceil)) \leq \sum_{\varphi \in \Phi: e \in \varphi} d(\text{OPT}'_\varphi) = \sum_{i \in \text{OPT}': e \in P(i)} d(i) \leq u(e).$$

It follows that $p(\text{APX}) \geq p(\text{APX}') \geq (1 - \varepsilon) \text{opt}'$. Using standard arguments, we conclude that

$$\begin{aligned} w(\text{APX}) &\geq \frac{\varepsilon \cdot w_{\max}}{n} \cdot p(\text{APX}) \\ &\geq (1 - \varepsilon) \cdot \frac{\varepsilon \cdot w_{\max}}{n} p(\text{OPT}') \\ &\geq (1 - \varepsilon) \cdot \frac{\varepsilon \cdot w_{\max}}{n} \cdot p(\text{OPT}) \\ &\geq (1 - \varepsilon) \cdot \left(\frac{\varepsilon \cdot w_{\max}}{n} \left(\frac{n}{\varepsilon \cdot w_{\max}} \cdot w(\text{OPT}) - n \right) \right) \\ &= (1 - \varepsilon) \cdot (\text{opt} - \varepsilon \cdot w_{\max}) \geq (1 - \varepsilon) \cdot (1 - \varepsilon) \text{opt}. \end{aligned} \quad \blacktriangleleft$$

It remains to upper bound the running time. Let **iters** be the number of iterations of the inner loop in Algorithm 3, i.e. the number of possible valid combinations for $(X_\varphi)_{\varphi \in \Phi}$. The bound on the running time follows easily from the following technical lemma.

► **Lemma 24.** $\text{iters} \leq \left(\frac{1+2\varepsilon}{\varepsilon} \cdot e \right)^{|\Phi|}$.

► **Lemma 25.** *Algorithm 3 runs in time $O\left(\frac{n^3}{\varepsilon} + \left(\frac{1}{\varepsilon}\right)^{O(m^2)} \cdot m^3 \cdot \log n\right)$.*

Proof. We already argued that the dynamic tables can be computed in total time $O\left(\frac{n^3}{\varepsilon}\right)$. We also observed that the outer for loop is executed $O\left(\frac{1}{\varepsilon} \log n\right)$ times. As already discussed, lines 6-8 take $O(|\Phi| \cdot m)$ time. Thus the second phase of the algorithm can be implemented in time $O\left(n^2 + |\Phi| \cdot m \cdot \text{iters} \cdot \frac{1}{\varepsilon} \cdot \log n\right)$ time. By Lemma 24, the overall running time of the algorithm is

$$O\left(\frac{n^3}{\varepsilon} + \left(\frac{1+2\varepsilon}{\varepsilon} \cdot e\right)^{|\Phi|} \cdot m \cdot |\Phi| \cdot \frac{1}{\varepsilon} \log n\right).$$

The claim follows since $|\Phi| \leq \frac{1}{2} \cdot m \cdot (m + 1)$. ◀

5:18 Unsplittable Flow on a Short Path

It remains to prove Lemma 24. To that aim, we need a standard bound on the binomial coefficients. Let $\mathcal{H}(x) = -x \cdot \ln(x) - (1-x) \cdot \ln(1-x)$ be the entropy function and assume $\mathcal{H}(0) = \mathcal{H}(1) = 0$.

► **Lemma 26** (Example 11.1.3 in [20]). *For every $n \in \mathbb{N}$ and integer $0 \leq k \leq n$ it holds that $\binom{n}{k} \leq \exp\left(n \cdot \mathcal{H}\left(\frac{k}{n}\right)\right)$.*

Proof of Lemma 24. Recall that `iters` is equal to the possible sequences of $|\Phi|$ non-negative integers whose sum is at most $Y = \lfloor \frac{1+\varepsilon}{\varepsilon} \cdot |\Phi| \rfloor$. These sequences can be represented via a binary string as follows. Let $\varphi_1, \dots, \varphi_{|\Phi|}$ be an arbitrary ordering of Φ , and $X_i = X_{\varphi_i}$. The bit string consists of X_1 many 1s, followed by one 0, followed by X_2 many 1s and so on, ending with the $X_{|\Phi|}$ many 1s, an additional 0 and a final padding of 1s till the target length of Y is reached. In particular all valid sequences correspond to binary strings with $Y + |\Phi|$ digits and exactly $|\Phi|$ zeros. It is therefore sufficient to upper bound the number of the latter bit strings, namely $\binom{Y+|\Phi|}{|\Phi|}$. By Lemma 26 we have,

$$\begin{aligned} \text{iters} &= \binom{|\Phi| + \lfloor \frac{1+\varepsilon}{\varepsilon} \cdot |\Phi| \rfloor}{|\Phi|} \\ &\leq \exp\left(\left(|\Phi| + \left\lfloor \frac{1+\varepsilon}{\varepsilon} \cdot |\Phi| \right\rfloor\right) \cdot \mathcal{H}\left(\frac{|\Phi|}{|\Phi| + \lfloor \frac{1+\varepsilon}{\varepsilon} \cdot |\Phi| \rfloor}\right)\right) \\ &\leq \exp\left(\left(|\Phi| \cdot \frac{1+2\varepsilon}{\varepsilon}\right) \cdot \mathcal{H}\left(\frac{|\Phi|}{|\Phi| \cdot \frac{1+2\varepsilon}{\varepsilon}}\right)\right) = \left(\exp\left(\frac{1+2\varepsilon}{\varepsilon} \cdot \mathcal{H}\left(\frac{\varepsilon}{1+2\varepsilon}\right)\right)\right)^{|\Phi|}, \end{aligned} \quad (17)$$

where the last inequality holds since $x \cdot \mathcal{H}\left(\frac{a}{x}\right)$ is increasing in x for any $a \geq 1$ and $|\Phi| + \lfloor \frac{1+\varepsilon}{\varepsilon} \cdot |\Phi| \rfloor \leq |\Phi| \cdot \frac{1+2\varepsilon}{\varepsilon}$. It also holds that

$$\begin{aligned} \frac{1+2\varepsilon}{\varepsilon} \cdot \mathcal{H}\left(\frac{\varepsilon}{1+2\varepsilon}\right) &= \frac{1+2\varepsilon}{\varepsilon} \cdot \left(-\frac{\varepsilon}{1+2\varepsilon} \cdot \ln\left(\frac{\varepsilon}{1+2\varepsilon}\right) - \left(1 - \frac{\varepsilon}{1+2\varepsilon}\right) \cdot \ln\left(1 - \frac{\varepsilon}{1+2\varepsilon}\right)\right) \\ &\leq -\ln\left(\frac{\varepsilon}{1+2\varepsilon}\right) - \frac{1+2\varepsilon}{\varepsilon} \cdot \left(1 - \frac{\varepsilon}{1+2\varepsilon}\right) \cdot \left(-\frac{\varepsilon}{1+2\varepsilon} \cdot \left(1 + \frac{\varepsilon}{1+2\varepsilon}\right)\right) \\ &= \ln\left(\frac{1+2\varepsilon}{\varepsilon}\right) + \left(1 - \frac{\varepsilon}{1+2\varepsilon}\right) \cdot \left(1 + \frac{\varepsilon}{1+2\varepsilon}\right) \\ &\leq \ln\left(\frac{1+2\varepsilon}{\varepsilon}\right) + 1 \end{aligned} \quad (18)$$

where the first inequality follows from $\ln(1-x) \geq -x(1+x)$ for $x \in (0,0.1)$, and the second inequality holds as $(1+x)(1-x) \leq 1$ for every $x \in \mathbb{R}$. By (17) and (18) we have,

$$\text{iters} \leq \left(\exp\left(\frac{1+2\varepsilon}{\varepsilon} \cdot \mathcal{H}\left(\frac{\varepsilon}{1+2\varepsilon}\right)\right)\right)^{|\Phi|} \leq \left(\exp\left(\ln\left(\frac{1+2\varepsilon}{\varepsilon}\right) + 1\right)\right)^{|\Phi|} = \left(\frac{1+2\varepsilon}{\varepsilon} \cdot e\right)^{|\Phi|} \blacktriangleleft$$

We now have the tools required to complete the proof of Theorem 5.

Proof of Theorem 5. It follows directly from Lemmas 22, 23 and 25 by choosing the parameter $\varepsilon/2$ so as to have a $(1-\varepsilon)$ approximation. \blacktriangleleft

4 A Lower bound for BagUFP

In this section we prove Theorem 1 using a simple reduction from the partition problem.

Proof of Theorem 1

Recall that in the NP -complete Partition problem we are given a collection of n non-negative integers $A = \{a_1, \dots, a_n\}$ in $[0, 1]$ whose sum is $2M$. Our goal is to determine whether there exists a subset of numbers whose sum is precisely M .

We show that an FPTAS for BagUFP in the considered case implies a polynomial time algorithm to solve Partition, hence the claim. We build (in polynomial time) an instance of BagUFP with 2 edges e_1 and e_2 , both of capacity M . Furthermore, for each a_j , we create two tasks t_j^1 and t_j^2 , with demand a_j and subpath e_1 and e_2 , resp. All the tasks have profit 1. The bags are given by the pairs $\{t_j^1, t_j^2\}$, $j = 1, \dots, n$. Obviously, the input Partition instance is a YES instance iff the optimal solution to the corresponding BagUFP instance has value n , i.e. exactly one task per bag is selected (notice that a solution cannot have larger profit). Indeed, given a solution $A' \subseteq A$ for the Partition instance, a valid solution to the corresponding BagUFP instance is obtained by selecting all the tasks t_j^1 with $j \in A'$ and all the tasks t_j^2 with $j \notin A'$. Notice that the total demand of the tasks using e_1 and e_2 must be exactly M . Vice versa, given a BagUFP solution S of profit n , the selected tasks $S_1 \subseteq S$ of type t_j^1 must have total demand exactly M , hence inducing a valid Partition solution $A' := \{j \in \{1, \dots, n\} : t_j^1 \in S_1\}$.

We run the mentioned FPTAS on the obtained BagUFP instance with parameter $\varepsilon = \frac{1}{2n}$ (hence taking polynomial time). If the optimal solution is n , the FPTAS will return a solution of profit at least $\frac{n}{1+\varepsilon} \geq n - \frac{1}{2+1/n} > n - 1$, hence a solution of profit n since the profit is an integer. Otherwise, the FPTAS will return a solution of profit at most $n - 1$. This is sufficient to discriminate between YES and NO instances of Partition. ◀

References

- 1 Hesham K Alfares. Survey, categorization, and comparison of recent tour scheduling literature. *Annals of Operations Research*, 127:145–175, 2004. doi:10.1023/B:ANOR.0000019088.98647.E2.
- 2 Aris Anagnostopoulos, Fabrizio Grandoni, Stefano Leonardi, and Andreas Wiese. Constant integrality gap LP formulations of unsplittable flow on a path. In *IPCO*, pages 25–36, 2013. doi:10.1007/978-3-642-36694-9_3.
- 3 Aris Anagnostopoulos, Fabrizio Grandoni, Stefano Leonardi, and Andreas Wiese. A mazing $2+\varepsilon$ approximation for unsplittable flow on a path. In *SODA*, pages 26–41, 2014.
- 4 Kenneth R Baker. Workforce allocation in cyclical scheduling problems: A survey. *Journal of the Operational Research Society*, 27(1):155–167, 1976.
- 5 N. Bansal, A. Chakrabarti, A. Epstein, and B. Schieber. A quasi-PTAS for unsplittable flow on line graphs. In *STOC*, pages 721–729. ACM, 2006. doi:10.1145/1132516.1132617.
- 6 N. Bansal, Z. Friggstad, R. Khandekar, and R. Salavatipour. A logarithmic approximation for unsplittable flow on line graphs. In *SODA*, pages 702–709, 2009.
- 7 A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC '00)*, pages 735–744. ACM, 2000. doi:10.1145/335305.335410.
- 8 R. Bar-Yehuda, M. Beder, Y. Cohen, and D. Rawitz. Resource allocation in bounded degree trees. In *ESA*, pages 64–75, 2006.

- 9 Jatin Batra, Naveen Garg, Amit Kumar, Tobias Mömke, and Andreas Wiese. New approximation schemes for unsplittable flow on a path. In *SODA*, pages 47–58, 2015. doi:10.1137/1.9781611973730.5.
- 10 Stephen E Bechtold, Michael J Brusco, and Michael J Showalter. A comparative evaluation of labor tour scheduling methods. *Decision Sciences*, 22(4):683–699, 1991.
- 11 Paul Bonsma, Jens Schulz, and Andreas Wiese. A constant-factor approximation algorithm for unsplittable flow on paths. *SIAM Journal on Computing*, 43:767–799, 2014. doi:10.1137/120868360.
- 12 Gruia Călinescu, Amit Chakrabarti, Howard J. Karloff, and Yuval Rabani. An improved approximation algorithm for resource allocation. *ACM Transactions on Algorithms*, 7:48:1–48:7, 2011. doi:10.1145/2000807.2000816.
- 13 Venkatesan T. Chakaravarthy, Anamitra R. Choudhury, Shalmoli Gupta, Sambuddha Roy, and Yogish Sabharwal. Improved algorithms for resource allocation under varying capacity. In *ESA*, pages 222–234, 2014. doi:10.1007/978-3-662-44777-2_19.
- 14 A. Chakrabarti, C. Chekuri, A. Gupta, and A. Kumar. Approximation algorithms for the unsplittable flow problem. *Algorithmica*, 47:53–78, 2007. doi:10.1007/S00453-006-1210-5.
- 15 C. Chekuri, A. Ene, and N. Korula. Unsplittable flow in paths and trees and column-restricted packing integer programs. In *APPROX-RANDOM*, pages 42–55, 2009.
- 16 C. Chekuri, M. Mydlarz, and F. Shepherd. Multicommodity demand flow in a tree and packing integer programs. *ACM Transactions on Algorithms*, 3, 2007.
- 17 B. Chen, R. Hassin, and M. Tzur. Allocation of bandwidth and storage. *IIE Transactions*, 34:501–507, 2002.
- 18 M. Chrobak, G. Woeginger, K. Makino, and H. Xu. Caching is hard, even in the fault model. In *ESA*, pages 195–206, 2010.
- 19 Marek Chrobak, Gerhard J Woeginger, Kazuhisa Makino, and Haifeng Xu. Caching is hard—even in the fault model. *Algorithmica*, 63(4):781–794, 2012.
- 20 TM Cover and Joy A Thomas. Elements of information theory, 2006.
- 21 A. Darmann, U. Pfersch, and J. Schauer. Resource allocation with time intervals. *Theoretical Computer Science*, 411:4217–4234, 2010. doi:10.1016/J.TCS.2010.08.028.
- 22 Syamantak Das and Andreas Wiese. On minimizing the makespan with bag constraints. In *13th Workshop on Models and Algorithms for Planning and Scheduling Problems*, page 186, 2017.
- 23 Ilan Doron-Arad, Fabrizio Grandoni, and Ariel Kulik. Unsplittable flow on a short path. *arXiv preprint*, 2024. doi:10.48550/arXiv.2407.10138.
- 24 Ilan Doron-Arad, Ariel Kulik, and Hadas Shachnai. An aptas for bin packing with clique-graph conflicts. In *Algorithms and Data Structures: 17th International Symposium, WADS 2021, Virtual Event, August 9–11, 2021, Proceedings 17*, pages 286–299. Springer, 2021. doi:10.1007/978-3-030-83508-8_21.
- 25 Ilan Doron-Arad, Ariel Kulik, and Hadas Shachnai. An aptas for bin packing with partition matroid via a new method for lp rounding. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2023)*. Schloss-Dagstuhl – Leibniz Zentrum für Informatik, 2023.
- 26 Ilan Doron-Arad, Ariel Kulik, and Hadas Shachnai. Budgeted matroid maximization: a parameterized viewpoint. *IPEC*, 2023.
- 27 Ilan Doron-Arad, Ariel Kulik, and Hadas Shachnai. An EPTAS for budgeted matching and budgeted matroid intersection via representative sets. In *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2023.
- 28 Ilan Doron-Arad, Ariel Kulik, and Hadas Shachnai. An EPTAS for budgeted matroid independent set. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 69–83, 2023. doi:10.1137/1.9781611977585.CH7.

- 29 Ilan Doron-Arad, Ariel Kulik, and Hadas Shachnai. An fptas for budgeted laminar matroid independent set. *Operations Research Letters*, 51(6):632–637, 2023. doi:10.1016/J.ORL.2023.10.005.
- 30 Ilan Doron-Arad, Ariel Kulik, and Hadas Shachnai. Lower bounds for matroid optimization problems with a linear constraint. *ICALP proc.*, 2024.
- 31 Ilan Doron-Arad and Hadas Shachnai. Approximating bin packing with conflict graphs via maximization techniques. In Daniël Paulusma and Bernard Ries, editors, *WG 2023, Fribourg, Switzerland, June 28-30, 2023, Revised Selected Papers*, 2023.
- 32 Kilian Grafe, Klaus Jansen, and Kim-Manuel Klein. An eptas for machine scheduling with bag-constraints. In *The 31st ACM Symposium on Parallelism in Algorithms and Architectures*, pages 135–144, 2019. doi:10.1145/3323165.3323192.
- 33 Fabrizio Grandoni, Salvatore Ingala, and Sumedha Uniyal. Improved approximation algorithms for unsplittable flow on a path with time windows. In *WAOA*, pages 13–24, 2015. doi:10.1007/978-3-319-28684-6_2.
- 34 Fabrizio Grandoni, Tobias Mömke, and Andreas Wiese. Faster $(1+\epsilon)$ -approximation for unsplittable flow on a path via resource augmentation and back. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPICs*, pages 49:1–49:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ESA.2021.49.
- 35 Fabrizio Grandoni, Tobias Mömke, and Andreas Wiese. A PTAS for unsplittable flow on a path. In *STOC*, pages 289–302. ACM, 2022. doi:10.1145/3519935.3519959.
- 36 Fabrizio Grandoni, Tobias Mömke, and Andreas Wiese. Unsplittable flow on a path: The game! In *SODA*, pages 906–926. SIAM, 2022. doi:10.1137/1.9781611977073.39.
- 37 Fabrizio Grandoni, Tobias Mömke, Andreas Wiese, and Hang Zhou. To augment or not to augment: Solving unsplittable flow on a path by creating slack. In *SODA*, pages 2411–2422, 2017. doi:10.1137/1.9781611974782.159.
- 38 Fabrizio Grandoni, Tobias Mömke, Andreas Wiese, and Hang Zhou. A $(5/3 + \epsilon)$ -approximation for unsplittable flow on a path: placing small tasks into boxes. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 607–619, 2018. doi:10.1145/3188745.3188894.
- 39 Fabrizio Grandoni and Rico Zenklusen. Approximation schemes for multi-budgeted independence systems. In *European Symposium on Algorithms*, pages 536–548. Springer, 2010. doi:10.1007/978-3-642-15775-2_46.
- 40 Hans Kellerer, Ulrich Pferschy, David Pisinger, Hans Kellerer, Ulrich Pferschy, and David Pisinger. The multiple-choice knapsack problem. *Knapsack problems*, pages 317–347, 2004.
- 41 S. Leonardi, A. Marchetti-Spaccamela, and A. Vitaletti. Approximation algorithms for bandwidth and storage allocation problems under real time constraints. In *FSTTCS*, pages 409–420, 2000.
- 42 Dániel Marx. Parameterized complexity and approximation algorithms. *Comput. J.*, 51(1):60–78, 2008. doi:10.1093/COMJNL/BXM048.
- 43 C. A. Phillips, R. N. Uma, and J. Wein. Off-line admission control for general scheduling problems. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '00)*, pages 879–888. ACM, 2000.
- 44 David Pisinger. A minimal algorithm for the multiple-choice knapsack problem. *European Journal of Operational Research*, 83(2):394–410, 1995.
- 45 Alexander Schrijver et al. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.
- 46 Prabhakant Sinha and Andris A Zoltners. The multiple-choice knapsack problem. *Operations Research*, 27(3):503–515, 1979. doi:10.1287/OPRE.27.3.503.
- 47 Frits C. R. Spieksma. On the approximability of an interval scheduling problem. *Journal of Scheduling*, 2:215–227, 1999.

5:22 Unsplittable Flow on a Short Path

- 48 Jorne Van den Bergh, Jeroen Beliën, Philippe De Bruecker, Erik Demeulemeester, and Liesje De Boeck. Personnel scheduling: A literature review. *European journal of operational research*, 226(3):367–385, 2013. doi:10.1016/J.EJOR.2012.11.029.
- 49 Andreas Wiese. A $(1+\epsilon)$ -approximation for unsplittable flow on a path in fixed-parameter running time. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPICs*, pages 67:1–67:13, 2017.




On Equivalence of Parameterized Inapproximability of k -Median, k -Max-Coverage, and 2-CSP

Karthik C. S.   

Rutgers University, Piscataway, NJ, USA

Euiwoong Lee   

University of Michigan, Ann Arbor, MI, USA

Pasin Manurangsi   

Google Research, Bangkok, Thailand

Abstract

Parameterized Inapproximability Hypothesis (PIH) is a central question in the field of parameterized complexity. PIH asserts that given as input a 2-CSP on k variables and alphabet size n , it is $W[1]$ -hard parameterized by k to distinguish if the input is perfectly satisfiable or if every assignment to the input violates 1% of the constraints.

An important implication of PIH is that it yields the tight parameterized inapproximability of the k -maxcoverage problem. In the k -maxcoverage problem, we are given as input a set system, a threshold $\tau > 0$, and a parameter k and the goal is to determine if there exist k sets in the input whose union is at least τ fraction of the entire universe. PIH is known to imply that it is $W[1]$ -hard parameterized by k to distinguish if there are k input sets whose union is at least τ fraction of the universe or if the union of every k input sets is not much larger than $\tau \cdot (1 - \frac{1}{e})$ fraction of the universe.

In this work we present a gap preserving FPT reduction (in the reverse direction) from the k -maxcoverage problem to the aforementioned 2-CSP problem, thus showing that the assertion that approximating the k -maxcoverage problem to some constant factor is $W[1]$ -hard implies PIH. In addition, we present a gap preserving FPT reduction from the k -median problem (in general metrics) to the k -maxcoverage problem, further highlighting the power of gap preserving FPT reductions over classical gap preserving polynomial time reductions.

2012 ACM Subject Classification Theory of computation \rightarrow Problems, reductions and completeness; Theory of computation \rightarrow W hierarchy

Keywords and phrases Parameterized complexity, Hardness of Approximation, Parameterized Inapproximability Hypothesis, max coverage, k-median

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.6

Related Version *Full Version*: <https://arxiv.org/abs/2407.08917>

Funding *Karthik C. S.*: This work was supported by the National Science Foundation under Grant CCF-2313372 and by the Simons Foundation, Grant Number 825876, Awardee Thu D. Nguyen.

Euiwoong Lee: Supported in part by NSF grant CCF-2236669 and Google.

Pasin Manurangsi: Part of this work was done while the author was visiting Rutgers University.

Acknowledgements We are grateful to the Dagstuhl Seminar 23291 for a special collaboration opportunity. We thank Vincent Cohen-Addad, Venkatesan Guruswami, Jason Li, Bingkai Lin, and Xuandi Ren for discussions.

1 Introduction

Approximation Algorithms and Fixed Parameter Tractability are two popular ways to cope with NP-hardness of computational problems. In recent years, there is a steady rise in results contributing to the theory of parameterized inapproximability (see e.g. [22]). These



© Karthik C. S., Euiwoong Lee, and Pasin Manurangsi;
licensed under Creative Commons License CC-BY 4.0

19th International Symposium on Parameterized and Exact Computation (IPEC 2024).

Editors: Édouard Bonnet and Paweł Rzażewski; Article No. 6; pp. 6:1–6:18



Leibniz International Proceedings in Informatics

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

include parameterized inapproximability results for the k -Set Intersection problem [38, 6], k -Set Cover problem [10, 31, 39, 33, 42], k -Clique problem [40, 30, 9], Steiner Orientation problem [47], fundamental problems in coding theory and lattice theory [4, 3], and more.

A major open question in the theory of parameterized inapproximability is the resolution of the *parameterized inapproximability hypothesis* (PIH) [43]. PIH asserts that there exists some $\varepsilon > 0$, such that it is $W[1]$ -hard to distinguish if a 2-CSP instance on k variables (k is the parameter) and alphabet size n is completely satisfiable or if every assignment to its variables satisfies at most $1 - \varepsilon$ fraction of the constraints. The analogue of (the resolution of) PIH in the NP-world is the celebrated PCP theorem [2, 1, 17], and thus positively resolving PIH is also appropriately dubbed as proving the “PCP theorem for Parameterized Complexity” [43].

PIH is known to imply the hardness of approximation (to some positive constant factor) of many fundamental problems in parameterized complexity (for which we do not know how to prove unconditional constant factor $W[1]$ -hardness), such as k -maxcoverage [14] (and consequently clustering problems such as minimizing the k -median and k -means objectives; see [14]), Directed Odd Cycle Transversal [43], Strongly Connected Steiner Subgraph [11], Planar Steiner Orientation [12], Grid Tiling and Determinant Maximization [46], Independent Set in H -free graphs [19], etc.

It is known that assuming the Gap Exponential Time Hypothesis (Gap-ETH) [45, 18] one can show that the gap problem on 2-CSP referred to in PIH is not in FPT (see e.g. [5]), and moreover that many parameterized inapproximability results discussed in this manuscript follow from Gap-ETH (e.g. [8, 44]). Moreover, in a recent breakthrough, Guruswami et al. [25], building on ideas in [41], proved that assuming the Exponential Time Hypothesis [27, 28], the aforementioned gap problem on 2-CSP is not in FPT. Subsequently, they even obtained near optimal conditional time lower bounds for the parameterized 2-CSP problem [24]. However, this paper is solely focused on parameterized inapproximability, and thus we will not further elaborate on the related works in fine-grained inapproximability.

A popular approach to make progress on central questions such as resolving PIH, is to prove unconditional hardness results for problems whose hardness is known only assuming PIH. Such an approach (of proving the implications unconditionally) has been historically very fruitful in complexity theory, for example, in the last decade it is in the attempt of proving improved unconditional NP-hardness of approximation result for the vertex cover problem [35] (whose optimal inapproximability is only known under Unique Games Conjecture [34]), that the 2-to-2 games theorem was proven [36]. Moreover, this approach is indeed an active line of research in the theory of parameterized inapproximability, leading to the hardness of approximation results for k -set cover problem [10, 31, 39], k -clique problem [40, 30, 9], and more.

The maximization version of the k -set cover problem, i.e., the k -maxcoverage problem is a fundamental optimization problem at the heart of many computation problems in computer science. For example, it is at the heart of many clustering problems [23]. Formally, in the k -maxcoverage problem we are given as input a pair $(\mathcal{U}, \mathcal{S})$ and a parameter k , where \mathcal{S} is a collection of sets over the universe \mathcal{U} , and the goal is to find k sets in \mathcal{S} whose union is of maximum cardinality. The k -maxcoverage problem is a canonical $W[2]$ -complete problem, and currently the $W[1]$ -hardness of approximating the k -maxcoverage problem to some constant factor only holds assuming PIH [14]. Thus, we ask:

Is it possible to prove constant inapproximability of the k -maxcoverage problem circumventing the resolution of PIH?

This question is particularly appealing since $W[1]$ -hardness of approximating k -set cover (minimization variant of k -maxcoverage) was established circumventing PIH [10, 31, 39], and more recently, similar progress was achieved for the k -clique problem as well [40, 30, 9].

Our first result is rather surprising (at least in first glance), that the answer to the above question is in the negative.

► **Theorem 1** (Informal statement; See Theorem 5 for a formal statement). *For every $\delta \in (0, 1/2]$ (where δ is allowed to depend on k), if approximating the k -maxcoverage problem to $(1 - \delta)$ factor is $W[1]$ -hard then approximating 2-CSP to $(1 - \frac{\delta^2}{4})$ factor is also $W[1]$ -hard (under randomized Turing reductions).*

Recall that in [14], the authors proved that for every $\varepsilon > 0$, assuming PIH, approximating the k -maxcoverage problem to a factor better than $1 - \frac{1}{e} + \varepsilon$ is $W[1]$ -hard (see Lemma 19 in [14]). Moreover, this inapproximability factor is tight [26]. Together with Theorem 1, we have that there is a gap preserving reduction in both directions¹ between 2-CSP (on k variables and alphabet size n) and the k -maxcoverage problem on $\text{poly}(n)$ sets over universe of size $\text{poly}(n)$.

To the best of our knowledge, we are not aware of a direct gap preserving reduction to the 2-CSP problem from the maxcoverage problem in the NP world, i.e., in other words we do not know how to directly prove the PCP theorem for NP, assuming that approximating maxcoverage to some constant factor is NP-hard. Furthermore, an interesting consequence of Theorem 1 is a gap amplification result for the k -maxcoverage problem in the parameterized complexity world: starting from the $W[1]$ -hardness of approximating the k -maxcoverage problem to $(1 - \varepsilon)$ factor (for some constant $\varepsilon > 0$), we can first apply Theorem 1 and then Lemma 19 of [14], to obtain that approximating the k -maxcoverage problem to $(1 - \frac{1}{e} + \varepsilon')$ factor, for any $\varepsilon' > 0$ is $W[1]$ -hard.

At a high level, the proof of Theorem 1 has three steps. Starting from an instance of the k -maxcoverage problem containing n sets, in the first step we subsample a universe of size $O_k(\log n)$ while retaining the gap in the completeness and soundness cases (Lemma 6). In the second step, we reduce from this new k -maxcoverage instance on the smaller universe to a variant of the 2-CSP instance called “Valued CSP” (see Section 2 for the definition) by first equipartitioning the universe into $O_k(1)$ many subuniverses and then constructing a Valued CSP instance where for a subset of variables, each variable in that subset is associated with a subuniverse and an assignment to that variable determines how each of the k solution sets cover this subuniverse. The rest of the variables (k many of them) encode the k solution sets (see Lemma 7). Finally, in the last step, we provide a gap preserving reduction from Valued CSP to the standard 2-CSP (Lemma 8).

Theorem 1 has further implications on our understanding of the complexity of approximating the k -maxcoverage problem. While exactly solving the problem is $W[2]$ -hard, it was known to experts that approximating the k -maxcoverage problem to $1 - \frac{1}{F(k)}$ factor, for any computable function F , is in $W[1]$. A corollary of Theorem 1 is a formal proof of this $W[1]$ -membership (by setting $\delta = 1/F(k)$ in Theorem 1). Moreover, by modifying the range of parameters in the reduction of [31] for the k -set cover instance, it is possible to argue that approximating the k -maxcoverage problem to any $1 - 1/\rho(k)$ factor is $W[1]$ -hard for every unbounded computable function ρ (for example think of $\rho(k) = \log^*(k)$). Thus, we obtain the following.

► **Theorem 2.** *Let $\rho : \mathbb{N} \rightarrow \mathbb{N}$ be any unbounded computable function. Then approximating the k -maxcoverage problem to $1 - \frac{1}{\rho(k)}$ factor is $W[1]$ -complete.*

¹ Albeit the reduction from the k -maxcoverage problem to the 2-CSP problem is a randomized Turing reduction.

We can extend our line of inquiry and wonder if one can prove the parameterized inapproximability of clustering objectives such as k -median and k -means without proving anything about the inapproximability of the k -maxcoverage problem. We will restrict our attention here to the k -median problem, but our results extend to the k -means problem as well (see Remark 12).

In the k -median problem (in general metric), we are given as input a tuple $((V, d), C, \mathcal{F}, \tau)$ and a parameter k , where V is a finite set, and d is a distance function for all pairs of points in V respecting the triangle inequality (or more precisely (V, d) is a metric space), $C, \mathcal{F} \subseteq V$ and the goal is to determine if there exists $F \subseteq \mathcal{F}$ such that $|F| = k$ and $\text{cost}(C, F)$ is minimized, where $\text{cost}(C, F)$ is the sum of distances from every client in C to its closest facility in F (see Section 2 for a formal definition). The k -median problem is $W[2]$ -complete by a simple reduction from the k -maxcoverage problem, and currently the $W[1]$ -hardness of approximating k -median to some constant factor only holds assuming PIH [14]. Thus, we ask:

*Is it possible to prove constant inapproximability of k -median
circumventing the resolution of PIH?*

Our result again is rather surprising that the answer to the above question is also in the negative.

► **Theorem 3.** *For any constants $\alpha, \delta > 0$, if approximating the k -median problem to $(1 + \alpha + \delta)$ factor is $W[1]$ -hard then approximating the multicolored k -maxcoverage problem to $(1 - \frac{\alpha}{2})$ factor is also $W[1]$ -hard (under randomized Turing reductions). A similar reduction also holds from the k -means problem to the multicolored k -maxcoverage problem.*

In Remark 10 we discuss how to modify the proof of Theorem 1 to obtain the statement of Theorem 1 to hold even for the multicolored k -maxcoverage problem instead of the (un-multicolored) k -maxcoverage problem. Then we can put together the reduction in Theorem 3 with the modified Theorem 1 to obtain an FPT gap preserving reduction from the k -median problem to 2-CSP problem.

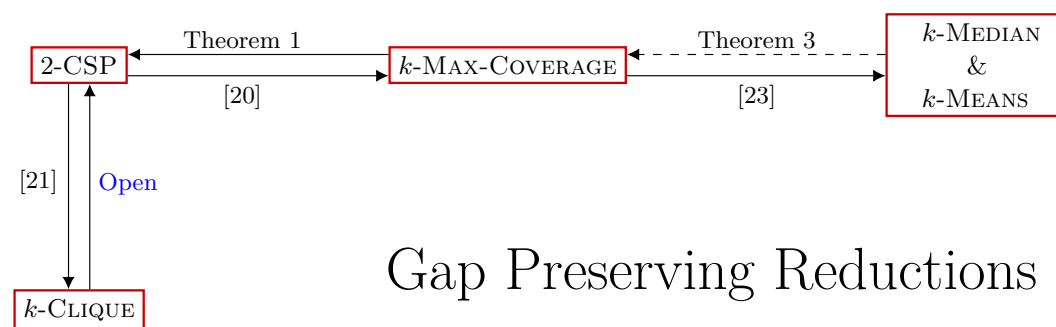
The proof of Theorem 3 follows from observing that the $(1 + \frac{\alpha}{\epsilon} + \epsilon)$ -approximation algorithm (for any $\epsilon > 0$) of [14] can be fine-tuned and rephrased as a reduction from k -median problem to the multicolored k -maxcoverage problem.

Both Theorems 1 and 3 illustrate the power of FPT gap preserving reductions over classical gap preserving polynomial time reductions. In particular, if we had a polynomial time analogue of Theorem 3, i.e., if the runtime of the algorithm \mathcal{A} and Γ in the Theorem 3 statement are both polynomial functions then this would lead to a major breakthrough in the field of approximation algorithms. In particular, it would show optimal approximation thresholds for the celebrated k -median and k -means problems in the NP world, improving on the state-of-the-art result of [15] and [29] respectively, showing that the hardness of approximation factors obtained in [23] are optimal!

In Figure 1, we highlight gap-preserving FPT reductions between k -clique, 2-CSP, k -maxcoverage, and k -median and k -means problems. A glaring open problem in Figure 1 is whether constant inapproximability of the k -clique problem implies PIH. This is a challenging open problem, even listed in [22].

2 Preliminaries

In this section, we formally define the problems of interest to this paper. Throughout, we use the notation $O_k(\cdot)$ and $\Omega_k(\cdot)$ to denote that the hidden constant can be any computable function of k .



■ **Figure 1** In the above figure, we provide bidirectional FPT gap preserving reductions between k -clique, 2-CSP, k -maxcoverage, and k -median and k -means problems, whenever possible, with appropriate references. The reduction from k -median and k -means problems is to the multicolored version of the k -maxcoverage problem (see Remark 13 for a discussion) and that is why the arrow is dashed.

k -maxcoverage problem

We denote by $(\mathcal{U}, \mathcal{S})$ a set system where \mathcal{U} denotes the universe and \mathcal{S} is a collection of subsets of \mathcal{U} . In the k -maxcoverage problem, we are given as input a set system $(\mathcal{U}, \mathcal{S})$ and a parameter k , and the goal is to identify k sets in \mathcal{S} whose union is of maximum size. We denote by $\text{OPT}(\mathcal{U}, \mathcal{S})$ the optimum fraction of the k -maxcoverage instance $(\mathcal{U}, \mathcal{S})$, i.e.

$$\max_{S_1, \dots, S_k \in \mathcal{S}} \frac{|S_1 \cup \dots \cup S_k|}{|\mathcal{U}|}.$$

We denote by $\text{GapMaxkCov}(\tau, \tau')$ the decision problem where given as input an instance $(\mathcal{U}, \mathcal{S})$ of the k -maxcoverage problem, the goal is to distinguish the completeness case where $\text{OPT}(\mathcal{U}, \mathcal{S}) \geq \tau$ and the soundness case where $\text{OPT}(\mathcal{U}, \mathcal{S}) < \tau'$. We also define $\text{GapMaxkCovSmallUni}(\tau, \tau')$ be the same problem but only restricted to the instances where $|\mathcal{U}| \leq O_k(\log |\mathcal{S}|)$.

In this paper, we also refer to the multicolored k -maxcoverage problem whose input is $(\mathcal{U}, \mathcal{S} := \mathcal{S}_1 \dot{\cup} \mathcal{S}_2 \dot{\cup} \dots \dot{\cup} \mathcal{S}_k)$, and the goal is to identify $(S_1, S_2, \dots, S_k) \in \mathcal{S}_1 \times \mathcal{S}_2 \times \dots \times \mathcal{S}_k$ such that $|S_1 \cup S_2 \cup \dots \cup S_k|$ is maximized. Moreover, we extend the above notations of OPT , GapMaxkCov , and $\text{GapMaxkCovSmallUni}$ to the multicolored k -maxcoverage problem.

2-CSP

For convenience, we use weighted version of 2-CSP where the edges are weighted. Note that there is a simple (FPT) reduction from this version to the unweighted version [16].

A 2-CSP instance $\Pi = (V, E, (\Sigma_v)_{v \in V}, (w_e)_{e \in E}, (C_e)_{e \in E})$ consists of the following:

- The set of vertices (i.e. variables) V .
- The set E of edges between V .
- For each $v \in V$, the alphabet set Σ_v of v .
- For each $e = (u, v) \in E$, a weight w_e and the constraint $C_e \subseteq \Sigma_u \times \Sigma_v$.

An *assignment* is a tuple $\psi = (\psi_v)_{v \in V}$ where $\psi_v \in \Sigma_v$. The (*weighted and normalized*) *value* of an assignment ψ , denoted by $\text{val}_\Pi(\psi)$, is defined as:

$$\frac{1}{\sum_{e \in E} w_e} \cdot \sum_{e=(u,v) \in E} w_e \cdot \mathbf{1}[(\psi_u, \psi_v) \in C_{u,v}],$$

6:6 On Parameterized Inapproximability of k -Median, k -Max-Coverage, and 2-CSP

where for any proposition Λ , $\mathbf{1}(\Lambda)$ is 1 if Λ is true and 0 otherwise. The value of the instance is defined as $\text{val}(\Pi) := \max_{\psi} \text{val}_{\Pi}(\psi)$ where the maximum is over all assignments ψ of Π . The *alphabet size* of the instance is defined as $\max_{v \in V} |\Sigma_v|$. If a 2-CSP instance is provided without $(w_e)_{e \in E}$ then the weights are all assumed to be 1.

The $\text{Gap2CSP}(c, s)$ problem is to decide whether a 2-CSP instance Π has value at least c or less than s . Note that the parameter of this problem is the number of variables in Π .

(Finite) Valued 2-CSP

It will also be more convenient for us to employ a more general version known as (*Finite*) *Valued CSP*². In short, this is a version where different assignments for each edge can result in different values. This is defined more precisely below.

- A Valued 2-CSP instance $\Pi = (V, E, (\Sigma_v)_{v \in V}, (f_e)_{e \in E})$ consists of
- $V, E, (\Sigma_v)_{v \in V}$ are defined similarly to 2-CSP instances.
 - For each $e = (u, v) \in E$, the value function $f_e : \Sigma_u \times \Sigma_v \rightarrow [0, 1]$.

The notion of an assignment is defined similar to 2-CSP instance but the value of an assignment is now defined as:

$$\text{val}_{\Pi}(\psi) := \mathbb{E}_{(u,v) \sim E} [f_{(u,v)}(\psi_u, \psi_v)].$$

The value of an instance is defined similar to before. We emphasize that the main difference between valued 2-CSP and standard 2-CSP is that the function f_e can take continuous value.

The $\text{Gap2VCSP}(c, s)$ problem is to decide whether a Valued 2CSP instance Π has value at least c or less than s . Again, the parameter here is the number of variables.

k -median

An instance of the k -median problem is defined by a tuple $((V, d), C, \mathcal{F}, k)$, where (V, d) is a metric space over a set of points V with $d(i, j)$ denoting the distance between two points i, j in V . Further, C and \mathcal{F} are subsets of V and are referred to as “clients” and “facility locations” respectively, and k is a positive parameter. The goal is to find a subset F of k facilities in \mathcal{F} to minimize

$$\text{cost}(C, F) := \sum_{j \in C} d(j, F),$$

where $d(j, F) := \min_{f \in F} d(j, f)$. The cost of the k -means objective is $\text{cost}_2(C, F) := \sum_{j \in C} d(j, F)^2$.

k -MaxCover problem

We recall the MaxCover problem introduced in [8]. A k -MaxCover instance Γ consists of a bipartite graph $G = (V \dot{\cup} W, E)$ such that V is partitioned into $V = V_1 \dot{\cup} \dots \dot{\cup} V_k$ and W is partitioned into $W = W_1 \dot{\cup} \dots \dot{\cup} W_{\ell}$. We sometimes refer to V_i 's and W_j 's as *left super-nodes* and *right super-nodes* of Γ , respectively.

A solution to k -MaxCover is called a *labeling*, which is a subset of vertices $v_1 \in V_1, \dots, v_k \in V_k$. We say that a labeling v_1, \dots, v_k *covers* a right super-node W_i , if there exists a vertex $w_i \in W_i$ which is a joint neighbor of all v_1, \dots, v_k , i.e., $(v_j, w_i) \in E$ for every $j \in [k]$. We denote by $\text{MaxCover}(\Gamma)$ the maximal fraction of right super-nodes that can be simultaneously covered, i.e.,

² See e.g. [37] and references therein.

$$\text{MaxCover}(\Gamma) = \frac{1}{\ell} \left(\max_{\text{labeling } v_1, \dots, v_k} \left| \{i \in [\ell] \mid W_i \text{ is covered by } v_1, \dots, v_k\} \right| \right).$$

Given an instance $\Gamma(G, c, s)$ of the k -MaxCover problem as input, our goal is to distinguish between the two cases:

Completeness $\text{MaxCover}(\Gamma) \geq c$.

Soundness $\text{MaxCover}(\Gamma) \leq s$.

Concentration Inequalities

We will also use the multiplicative Chernoff inequality, which is summarized below.

► **Theorem 4** (Chernoff Inequality). *Let X_1, \dots, X_m denote i.i.d. Bernoulli random variables where $\mathbb{E}[X_j] = q$. Then, for any $\zeta \in (0, 1)$ we have*

$$\Pr[X_1 + \dots + X_m \geq (1 + \zeta)qm], \Pr[X_1 + \dots + X_m \leq (1 - \zeta)qm] \leq \exp(-\zeta^2 qm/3).$$

3 Reducing k -MaxCoverage to 2-CSP

In this section, we prove the following formal version of Theorem 1.

► **Theorem 5.** *For every $\tau > 0$ and $\delta \in (0, 1/2]$ (where both τ and δ are allowed to depend on k), there is a randomized algorithm \mathcal{A} which takes as input a k -maxcoverage instance $(\mathcal{U}, \mathcal{S})$ and with probability $1 - o(1)$, outputs $\Gamma(k)$ many 2-CSP instances $\{\Pi_i\}_{i \in [\Gamma(k)]}$, for some computable function $\Gamma: \mathbb{N} \rightarrow \mathbb{N}$, such that the following holds.*

Running Time: \mathcal{A} runs in time $T(k) \cdot \text{poly}(|\mathcal{U}| + |\mathcal{S}|)$, for some computable function $T: \mathbb{N} \rightarrow \mathbb{N}$.

Size: For every $i \in [\Gamma(k)]$, we have that Π_i is defined on $\Lambda(k)$ variables over an alphabet of size $\text{poly}(|\mathcal{U}| + |\mathcal{S}|)$ for some computable function $\Lambda: \mathbb{N} \rightarrow \mathbb{N}$.

Completeness: Suppose there exist k sets in \mathcal{S} such that their union is of size $\tau \cdot |\mathcal{U}|$. Then, there exists $i \in [\Gamma(k)]$ and an assignment to the variables of Π_i that satisfies all its constraints.

Soundness: Suppose that for every k sets in \mathcal{S} , their union is of size at most $(1 - \delta) \cdot \tau \cdot |\mathcal{U}|$. Then, for every $i \in [\Gamma(k)]$ we have that every assignment to the variables of Π_i satisfies at most $\left(1 - \frac{\delta^2}{4}\right)$ fraction of the constraints of Π_i .

The proof of the theorem follows in three steps. We start by providing a randomized reduction which shows that we may assume w.l.o.g. that $|\mathcal{U}| \leq O_k(\log n)$. The rough idea is to use random hashing and subsampling to reduce the domain, as formalized below.

► **Lemma 6.** *For every $\tau > 0$ and $\delta \in (0, 1/2]$ (where both τ, δ may or may not depend on k), there is a randomized FPT reduction (that holds w.p. $1 - o(1)$) from $\text{GapMaxkCov}(\tau, (1 - \delta)\tau)$ to $\text{GapMaxkCovSmallUni}(\tau', (1 - \varepsilon)\tau')$ for $\varepsilon = \delta^2/2$ and $\tau' = \delta(1 - \delta) \cdot (1 + \delta^2)$.*

In the second step, we show how to reduce the small-universe k -maxcoverage instance to a Valued CSP instance. The overall idea is to create a set of variables x_1, \dots, x_k where x_i represents the i -th set selected in the solution. To check that they cover a large number of constraints, we partition the universe into M groups $\mathcal{U}_1, \dots, \mathcal{U}_M$ each of size $O(\log n / \log k)$ where the small-universe k -maxcoverage instance guarantees that $M = O_k(1)$. For each partition $j \in [M]$, we create a variable y_j . The variable encodes how x_1, \dots, x_k covers the

j -th partition \mathcal{U}_j . Namely, each $\sigma \in \Sigma_{y_j}$ encodes whether each element (in \mathcal{U}_j) is covered and, if so, by which set. Notice that there can be as many as $(k+1)^{|\mathcal{U}_j|}$ possibilities here, but this is not an issue since $|\mathcal{U}_j| = O(\log n / \log k)$. The constraints are then simply the consistency checks between x_i, y_j where the values represents the number of elements the i -th set covers in \mathcal{U}_j . Formally, we prove the following.

► **Lemma 7.** *For every $\tau, \varepsilon > 0$ (where both τ, ε may or may not depend on k), there is a deterministic FPT reduction from $\text{GapMaxkCovSmallUni}(\tau, (1-\varepsilon)\tau)$ to $\text{Gap2VCSP}(c, c(1-\varepsilon))$ where $c = O_k(\tau)$.*

The final part is the following lemma which shows that, in the FPT world, Gap2VCSP reduces to Gap2CSP . At a high level, this reduction is done by guessing the values of each edge (in the optimal solution) and turning that into a “hard” constraint as in a 2-CSP.

► **Lemma 8.** *For any $c > s > 0$ such that $c/s \geq 1 + \Omega_k(1)$, there is a deterministic FPT (Turing) reduction from $\text{Gap2VCSP}(c, s)$ to $\text{Gap2CSP}(1, 1-\varepsilon)$ for $\varepsilon = \frac{c/s-1}{c/s+1}$.*

Finally, we put together the above three lemmas to prove Theorem 5.

Proof of Theorem 5. The algorithm \mathcal{A} takes as input an instance of $\text{GapMaxkCov}(\tau, (1-\delta)\tau)$, applies the reduction in Lemma 6 to obtain an instance of $\text{GapMaxkCovSmallUni}(\tau', (1-\frac{\delta^2}{2})\tau')$ for $\tau' = \delta(1-\delta) \cdot (1+\delta^2)$, and then applies the reduction in Lemma 7 to obtain an instance of $\text{Gap2VCSP}(c, c(1-\frac{\delta^2}{2}))$ where $c = O_k(\delta)$, and finally applies the reduction in Lemma 8 to obtain an instance of $\text{Gap2CSP}(1, 1-\varepsilon)$ for $\varepsilon = \frac{\delta^2}{4-\delta^2} > \frac{\delta^2}{4}$. ◀

3.1 Step I: Universe Reduction for k -MaxCoverage

In this subsection, we prove Lemma 6.

Proof of Lemma 6. Given $(\mathcal{U}, \mathcal{S} = \{S_1, \dots, S_n\})$, an instance of the $\text{GapMaxkCov}(\tau, (1-\delta)\tau)$ problem, we create an instance $(\mathcal{U}', \mathcal{S}' = \{S'_1, \dots, S'_n\})$ of the $\text{GapMaxkCovSmallUni}(\tau', (1-\varepsilon)\tau')$ problem as follows. Let $m := \lceil 12k \cdot \delta^{-9} \log n \rceil$ and $p := \frac{\delta}{\tau \cdot |\mathcal{U}|}$. Let $\mathcal{U}' = [m]$. For each $u \in \mathcal{U}$ and $j \in [m]$, let $Y_{u,j}$ denote an i.i.d. Bernoulli random variable that is 1 with probability p . Then, for each $i \in [n]$ and $j \in [m]$, let j belong to S'_i if and only if there exist some $u \in S_i$ such that $Y_{u,j} = 1$.

Fix any $S_{i_1}, \dots, S_{i_k} \in \mathcal{S}$. For every $j \in [m]$, let X_j denote the indicator whether $j \in S'_{i_1} \cup \dots \cup S'_{i_k}$. Note that X_1, \dots, X_m are i.i.d. and,

$$\Pr[X_j = 1] = 1 - (1-p)^{|S_{i_1} \cup \dots \cup S_{i_k}|}.$$

Note also that $X_1 + \dots + X_m$ is exactly equal to $|S'_{i_1} \cup \dots \cup S'_{i_k}|$.

Completeness. Suppose that $\text{OPT}(\mathcal{U}, \mathcal{S}) \geq \tau$. Let S_{i_1}, \dots, S_{i_k} be an optimal solution in $(\mathcal{U}, \mathcal{S})$. Then, we have for each $j \in [m]$ that:

$$\Pr[X_j = 1] \geq 1 - (1-p)^{\tau \cdot |\mathcal{U}|} \geq 1 - \frac{1}{(1+p)^{\tau \cdot |\mathcal{U}|}} \geq 1 - \frac{1}{1+p\tau|\mathcal{U}|} = \frac{\delta}{1+\delta} = \frac{1}{1-\delta^4} \cdot \tau',$$

where the second inequality follows from Bernoulli's inequality and the last inequality follows from our choice of parameters. Applying Theorem 4 with $\zeta = \delta^4$ implies that $\Pr[X_1 + \dots + X_m \geq \tau'] \geq 1 - \exp(-\delta^9 m/6) = 1 - o(1)$ as desired.

Soundness. Suppose that $\text{OPT}(\mathcal{U}, \mathcal{S}) < (1 - \delta)\tau$. Consider any $S_{i_1}, \dots, S_{i_k} \in \mathcal{S}$. We have for each $j \in [m]$ that:

$$\Pr[X_j = 1] \leq 1 - (1 - p)^{(1-\delta)\tau \cdot |\mathcal{U}|} \leq (1 - \delta)p\tau|\mathcal{U}| = \delta(1 - \delta) = \frac{1}{1 + \delta^2} \cdot \tau',$$

where the second inequality follows from Bernoulli's inequality. Again, applying Theorem 4 with $\zeta = \delta^4$ implies that $\Pr[X_1 + \dots + X_m \geq \frac{1+\delta^4}{1+\delta^2}\tau'] \leq \exp(-\delta^9(1 - \delta)m/3) \leq \exp(-\delta^9m/6)$ (where we used that $\delta \leq 1/2$). Taking the union bound over all i_1, \dots, i_k then implies that this holds for all i_1, \dots, i_k with probability at least $1 - \frac{n^k}{\exp(\delta^9m/6)} \geq 1 - \frac{1}{n^k} = 1 - o(1)$. ◀

► **Remark 9.** We note that we can mimic the above proof to extend Lemma 6 to the multicolored k -maxcoverage problem as well. In particular, this gives a randomized FPT reduction (that holds w.p. $1 - o(1)$) from multicolored $\text{GapMaxkCov}(\tau, (1 - \delta)\tau)$ to multicolored $\text{GapMaxkCovSmallUni}(\tau', (1 - \varepsilon)\tau')$ for $\varepsilon = \delta^2/2$ and $\tau' = \delta(1 - \delta) \cdot (1 + \delta^2)$ as well.

3.2 Step II: Small-Universal k -MaxCoverage \Rightarrow Valued CSP

In this subsection, we prove Lemma 7.

Proof of Lemma 7. Given an instance $(\mathcal{U}, \mathcal{S} = \{S_1, \dots, S_n\})$ of $\text{GapMaxkCovSmallUni}(\tau, (1 - \varepsilon)\tau)$, we construct an instance $\Pi = (V, E, (\Sigma_v)_{v \in V}, (f_e)_{e \in E})$ of $\text{Gap2VCSP}(c, c(1 - \varepsilon))$ as follows:

- Let $M := \left\lceil \frac{|\mathcal{U}|}{\log |\mathcal{S}|} \cdot \log k \right\rceil = O_k(1)$ and let $\mathcal{U}_1 \dot{\cup} \dots \dot{\cup} \mathcal{U}_M$ be a partition of \mathcal{U} into nearly equal parts, each of size $|\mathcal{U}_i| = O(|\mathcal{U}|/M) = O(\log n / \log k)$.
- Let $V = \{x_1, \dots, x_k, y_1, \dots, y_M\}$ and E contains (x_i, y_j) for all $i \in [k]$ and $j \in [M]$.
- For each $i \in [k]$, let $\Sigma_{x_i} = [n]$.
- For each $j \in [M]$, let Σ_{y_j} contains all functions from \mathcal{U}_j to $\{0, \dots, k\}$.
- For each $i \in [k], j \in [M]$, let $f_{(x_i, y_j)}$ be defined as follows:

$$f_{(x_i, y_j)}(\sigma_u, \sigma_v) = \begin{cases} \frac{|\sigma_v^{-1}(i)|}{|\mathcal{U}|} & \text{if } \sigma_v^{-1}(i) \subseteq S_{\sigma_u}, \\ 0 & \text{otherwise.} \end{cases}$$

- Finally, let $c = \frac{\tau}{k \cdot M}$.

Note that the new parameter is $k + M = O_k(1)$. Furthermore, the running time of the reduction is polynomial since $|\Sigma_{y_j}| = (k + 1)^{|\mathcal{U}_j|} = k^{O(\log n / \log k)} = n^{O(1)}$. Thus, the reduction is an FPT reduction as desired (where the parameter is the number of variables of the Gap2VCSP instance).

We next prove the completeness and soundness of the reduction. In fact, we will argue that $\text{val}(\Pi) = \frac{\text{OPT}(\mathcal{U}, \mathcal{S})}{k \cdot M}$, from which the completeness and soundness immediately follow. To see that $\text{val}(\Pi) \geq \frac{\text{OPT}(\mathcal{U}, \mathcal{S})}{k \cdot M}$, let $S_{\ell_1}, \dots, S_{\ell_k}$ denote an optimal solution. We let $\psi_{x_i} = \ell_i$ for all $i \in [k]$. As for ψ_{y_j} , we let $\psi_{y_j}(u)$ be 0 if $u \notin S_{\ell_1} \cup \dots \cup S_{\ell_k}$; otherwise, we let $\psi_{y_j}(u) = i$ such that S_{ℓ_i} (if there are multiple such i 's, just pick one arbitrarily). It is obvious by the construction that $\psi_{y_j}^{-1}(i) \subseteq S_{\psi_{x_i}}$ for all $i \in [k]$ and $j \in [M]$. Thus, we have

$$\begin{aligned} \text{val}(\Pi) \geq \text{val}_\Pi(\psi) &= \frac{1}{k \cdot M} \sum_{j \in [M]} \sum_{i \in [k]} \frac{|\psi_{y_j}^{-1}(i)|}{|\mathcal{U}|} = \frac{1}{k \cdot M} \sum_{j \in [M]} \frac{|\mathcal{U}_j \cap (S_{\ell_1} \cup \dots \cup S_{\ell_k})|}{|\mathcal{U}|} \\ &= \frac{1}{k \cdot M} \frac{|S_{\ell_1} \cup \dots \cup S_{\ell_k}|}{|\mathcal{U}|} = \frac{\text{OPT}(\mathcal{U}, \mathcal{S})}{k \cdot M}. \end{aligned}$$

6:10 On Parameterized Inapproximability of k -Median, k -Max-Coverage, and 2-CSP

On the other hand, to show that $\text{val}(\Pi) \leq \frac{\text{OPT}(\mathcal{U}, \mathcal{S})}{k \cdot M}$, let ψ be any assignment of Π . We have

$$\begin{aligned} \text{val}_\Pi(\psi) &\leq \frac{1}{k \cdot M} \sum_{j \in [M]} \sum_{i \in [k]} \frac{|\psi_{y_j}^{-1}(i) \cap S_{\psi_{x_i}}|}{|\mathcal{U}|} \\ &\leq \frac{1}{k \cdot M} \sum_{j \in [M]} \frac{|\mathcal{U}_j \cap (S_{\psi_{x_1}} \cup \dots \cup S_{\psi_{x_k}})|}{|\mathcal{U}|} \\ &\leq \frac{1}{k \cdot M} \frac{|S_{\psi_{x_1}} \cup \dots \cup S_{\psi_{x_k}}|}{|\mathcal{U}|} \leq \frac{\text{OPT}(\mathcal{U}, \mathcal{S})}{k \cdot M}. \quad \blacktriangleleft \end{aligned}$$

► **Remark 10.** We can extend Lemma 7 to apply for the multicolored k -maxcoverage problem as well in the following way. In particular, we can start from the multicolored $\text{GapMaxkCov}(\tau, (1 - \delta)\tau)$ problem and as described in Remark 9, we can reduce it to the multicolored $\text{GapMaxkCovSmallUni}(\tau', (1 - \varepsilon)\tau')$ problem for $\varepsilon = \delta^2/2$ and $\tau' = \delta(1 - \delta) \cdot (1 + \delta^2)$. Then, we note that we can mimic the proof of Lemma 7 with one minor modification that for all $i \in [k]$, the alphabet set of variable x_i are the indices of the i^{th} collection of the input sets instead of the entire set $[n]$.

3.3 Step III: Valued CSP \Rightarrow 2-CSP

In this subsection, we prove Lemma 8.

Proof of Lemma 8. Let $\Pi = (V, E, (\Sigma_v)_{v \in V}, (f_e)_{e \in E})$ be a Valued 2-CSP instance, and let $\ell = |E|$ and $\gamma = \ell \cdot s$. We may assume that $\text{supp}(f_e) \subseteq [0, \gamma]$ for all $e \in E$. Indeed, if any $\sigma_u \in \Sigma_u, \sigma_v \in \Sigma_v$ for $(u, v) \in E$ satisfies $f_{(u,v)}(\sigma_u, \sigma_v) \geq \ell \cdot s$, then assigning σ_u to u and σ_v to v alone already yields value at least s . We describe the reduction under this assumption.

Let $B = \lceil 2\ell/\varepsilon \rceil$. For each $\theta \in [B]^{|E|}$, check if $\frac{1}{|E|} \sum_{e \in E} \theta_e \geq B/\gamma \cdot \frac{s}{1 - \varepsilon}$. If not, then skip this θ and continue to the next one. Otherwise, if this is satisfied, we create an instance $\Pi^\theta = (V, E, (\Sigma_v)_{v \in V}, (w_e^\theta)_{e \in E}, (C_e^\theta)_{e \in E})$ as follows:

- $V, E, (\Sigma_v)_{v \in V}$ remains the same as in Π .
- For each $e = (u, v) \in E$, let $w_e^\theta = \theta_e$ and $C_e = \{(\sigma_u, \sigma_v) \mid f_e(\sigma_u, \sigma_v) \geq \gamma \cdot \theta_e/B\}$.

Note that the number of different θ 's is $B^{|E|} = O(\ell/\varepsilon)^\ell \leq 2^{O(\ell \log \ell)}$ and thus the above is an FPT reduction. We next prove the completeness and soundness of the reduction.

Completeness. Suppose that there is an assignment ψ of Π such that $\text{val}_\Pi(\psi) \geq c$. Let θ^ψ be defined by $\theta_e^\psi := \lfloor B \cdot f_e(\psi_u, \psi_v)/\gamma \rfloor$ for all $e = (u, v) \in E$. Notice that

$$\begin{aligned} \frac{1}{|E|} \sum_{e \in E} \theta_e^\psi &= \frac{1}{|E|} \sum_{e=(u,v) \in E} \lfloor B \cdot f_e(\psi_u, \psi_v)/\gamma \rfloor \\ &\geq \frac{1}{|E|} \sum_{e=(u,v) \in E} (B \cdot f_e(\psi_u, \psi_v)/\gamma - 1) \\ &= B/\gamma \cdot \text{val}_\Pi(\psi) - 1 \\ &\geq B/\gamma \cdot c - 1 \\ &\geq B/\gamma \cdot \frac{s}{1 - \varepsilon}, \end{aligned}$$

where the last inequality follows from our choice of B and ε .

Thus, the instance Π^ψ is considered in the construction. It is also obvious by the construction that Π^ψ is indeed satisfiable.

Soundness. Suppose (contrapositively) that for some θ with $\frac{1}{|E|} \sum_{e \in E} \theta_e \geq B/\gamma \cdot s$ such that Π^θ is not a NO instance of $\text{Gap2CSP}(1, 1 - \varepsilon)$. That is, there exists an assignment ψ such that $\text{val}_{\Pi^\theta}(\psi) \geq 1 - \varepsilon$. From this, we have

$$\begin{aligned} \text{val}_{\Pi}(\psi) &= \frac{1}{|E|} \sum_{e=(u,v) \in E} f_e(\psi_u, \psi_v) \\ &\geq \frac{1}{|E|} \sum_{e=(u,v) \in E} (\gamma \cdot \theta_e / B) \cdot \mathbf{1}[(\psi_u, \psi_v) \in C_e] \\ &= \frac{1}{|E|} \gamma / B \cdot \text{val}_{\Pi^\theta}(\psi) \cdot \left(\sum_{e \in E} \theta_e \right) \\ &\geq \frac{1}{|E|} \gamma / B \cdot (1 - \varepsilon) \cdot \left(B / \gamma \cdot \frac{s}{1 - \varepsilon} \right) \\ &\geq s, \end{aligned}$$

where the first inequality is based on how C_e^θ is defined and the second inequality follows from $\text{val}_{\Pi^\theta}(\psi) \geq 1 - \varepsilon$ and the assumption on θ .

Thus, in this case, we have $\text{val}(\Pi) \geq s$ as desired. \blacktriangleleft

4 Reducing k -median to Multicolored k -MaxCoverage

In this section, we prove the following formal version of Theorem 3.

► **Theorem 11.** *For every constant $\alpha, \delta > 0$, there is an algorithm \mathcal{A} which takes as input a k -median instance $((V, d), C, \mathcal{F}, \tau)$ and outputs $\Gamma(k)$ many multicolored k -maxcoverage instances $\{\mathcal{U}^i, \mathcal{S}^i := \mathcal{S}_1^i \cup \mathcal{S}_2^i \cup \dots \cup \mathcal{S}_k^i\}_{i \in [\Gamma(k)]}$, for some computable function $\Gamma : \mathbb{N} \rightarrow \mathbb{N}$, such that the following holds.*

Running Time: \mathcal{A} runs in time $T(k) \cdot \text{poly}(|V|)$, for some computable function $T : \mathbb{N} \rightarrow \mathbb{N}$.

Size: For every $i \in [\Gamma(k)]$, we have that $|\mathcal{U}^i|, |\mathcal{S}^i| = \text{poly}(|V| \cdot \Delta)$, where $\Delta := \frac{\max_{v, v' \in V} d(v, v')}{\min_{\substack{v, v' \in V \\ d(v, v') > 0}} d(v, v')}$.

Completeness: Suppose that there exist $F \subseteq \mathcal{F}$ such that $|F| = k$ and $\text{cost}(C, F) \leq \tau$ then there exists $i \in [\Gamma(k)]$ and $(S_1, S_2, \dots, S_k) \in \mathcal{S}_1^i \times \mathcal{S}_2^i \times \dots \times \mathcal{S}_k^i$ such that $S_1 \cup S_2 \cup \dots \cup S_k = \mathcal{U}^i$.

Soundness: Suppose that for every $F \subseteq \mathcal{F}$ such that $|F| = k$ we have $\text{cost}(C, F) \geq (1 + \alpha + \delta) \cdot \tau$ then for every $i \in [\Gamma(k)]$ and every $(S_1, S_2, \dots, S_k) \in \mathcal{S}_1^i \times \mathcal{S}_2^i \times \dots \times \mathcal{S}_k^i$ we have $|S_1 \cup S_2 \cup \dots \cup S_k| \leq (1 - \frac{\alpha}{2}) \cdot |\mathcal{U}^i|$.

A similar reduction also holds from the k -means problem to the multicolored k -maxcoverage problem.

Thus, from the above theorem we can show that a $(1 - 1/e - \varepsilon)$ -FPT approximation for the multicolored k -maxcoverage problem implies a $(1 + 2/e + 3\varepsilon)$ -FPT approximation for the k -median problem (by setting $\alpha = 2\varepsilon + (2/e)$ and $\delta = \varepsilon$ in Theorem 11). This reduction was almost established in Cohen-Addad et al. [13] who gave an $(1 + 2/e + \varepsilon)$ -approximation for k -median in time $(k \log k / \varepsilon)^{O(k)} \text{poly}(n)$, using a $(1 - 1/e)$ -approximation algorithm for *Monotone Submodular Maximization with a (Partition) Matroid Constraint*: given a monotone submodular function $f : U \rightarrow \mathbb{R}_{\geq 0}$ and a partition matroid (U, \mathcal{I}) , compute a set $S \in \mathcal{I}$ that maximizes $f(S)$. Here we observe that the multicolored k -maxcoverage problem can replace the general submodular maximization.

Algorithm of [13]

First, we summarize the key steps from [13] without proofs. First, they compute a *coreset* of size $O(k \log n / \varepsilon^2)$; it is a (weighted) subset of clients $\mathcal{C}' \subseteq \mathcal{C}$ such that for any solution $F \subseteq \mathcal{F}$ with $|F| = k$, the k -median costs for \mathcal{C}' and \mathcal{C} are within a $(1 + \varepsilon)$ factor of each other. Therefore, for the rest of the discussion, let \mathcal{C} be the coreset itself and assume $|\mathcal{C}| = O(k \log n / \varepsilon^2)$.

Let $F^* = \{f_1^*, \dots, f_k^*\}$ be the centers in the optimal solution, and C_i^* be the set of clients served by f_i^* in the optimal solution. For each $i \in [k]$, let $\ell_i \in C_i^*$ be the client closest to f_i^* (ties broken arbitrarily) and call it the *leader* of C_i^* .

By exhaustive enumerations, in $(k \log n / \varepsilon)^{O(k)}$ time (which is upper bounded by $(k \log k / \varepsilon)^{O(k)} \text{poly}(n)$ time), one can guess ℓ_1, \dots, ℓ_k as well as R_1, \dots, R_k such that $d(\ell_i, f_i^*) \leq [R_i / (1 + \varepsilon), R_i]$.

Let $F_i := \{f \in \mathcal{F} : d(f, \ell_i) \leq R_i\}$. (One can assume F_i 's are disjoint by duplicating facilities.)

At this point, opening an arbitrary $f_i \in F_i$ for each $i \in [k]$ ensures a $(3 + \varepsilon)$ -approximation, as each client $c \in C_i^*$ can be connected to f_i where

$$d(c, f_i) \leq d(c, f_i^*) + d(f_i^*, \ell_i) + d(\ell_i, f_i) \leq (3 + \varepsilon) \cdot d(c, f_i^*),$$

since $d(f_i^*, \ell_i) \leq d(c, f_i^*)$ follows from the definition of the leader and $d(\ell_i, f_i) \leq (1 + \varepsilon)d(\ell_i, f_i^*)$ by the definition of R_i and F_i .

To further improve the approximation ratio to $(1 + 2/e + \varepsilon)$, [13] defined the function $\text{improv} : \mathcal{P}(\mathcal{F}) \rightarrow \mathbb{R}_{\geq 0}$ as follows (here \mathcal{P} denotes the power set). First, for each $i \in [k]$, add a *fictitious facility* f'_i , whose distance to ℓ_i is R_i and the distances to the other points are determined by shortest paths through ℓ_i ; i.e., for any x , we define $d(f'_i, x) := R_i + d(\ell_i, x)$. Let $F' := \{f'_1, \dots, f'_k\}$. The above paragraph's reasoning again also shows that:

$$\text{cost}(C, F') \leq (3 + \varepsilon) \cdot \text{OPT}. \quad (1)$$

For $S \subseteq \mathcal{F}$, $\text{improv}(S) := \text{cost}(C, F') - \text{cost}(C, S \cup F')$. Since any $f \in F_i$ is at a distance at most R_i from ℓ_i , as long as $|S \cap F_i| \geq 1$ for every $i \in [k]$, we have $\text{cost}(C, S \cup F') = \text{cost}(C, S)$.

[13] proved that $\text{improv}(\cdot)$ is monotone and submodular, so one can use [7]'s algorithm which obtains an $(1 - 1/e)$ -approximation algorithm for Monotone Submodular Maximization with a Matroid Constraint to find S such that $|S \cap F_i| = 1$ for every $i \in [k]$ and $\text{improv}(S) \geq (1 - 1/e) \cdot \text{improv}(F^*)$, which implies that for an approximate solution S^* we have:

$$\begin{aligned} \text{cost}(C, S^*) &= \text{cost}(C, F') - \text{improv}(S^*) \\ &\leq \text{cost}(C, F') - (1 - 1/e) \cdot \text{improv}(F^*) \\ &\leq \text{cost}(C, F') - (1 - 1/e) \cdot (\text{cost}(C, F') - \text{OPT}) \\ &= (1 - 1/e) \cdot \text{OPT} + (1/e) \cdot \text{cost}(C, F') \\ &\leq (1 + 2/e + \varepsilon) \cdot \text{OPT}. \end{aligned} \quad (2)$$

 $\text{improv}(\cdot)$ as a Coverage Function

Since the matroid constraint exactly corresponds to the *multicolor* part of the multicolored k -maxcoverage problem, it suffices to show that $\text{improv}(\cdot)$ can be realized as a coverage function; it will imply that a $(1 - 1/e - \varepsilon)$ -approximation algorithm for multicolored k -maxcoverage problem will imply $(1 + 2/e + O(\varepsilon))$ -approximation for k -median in FPT time.

Actually, the structure of $\text{improv}(\cdot)$ as the cost difference between two k -median solutions makes it easy to do so. Let us do it for a *weighted* coverage function where each element e has weight $w(e)$ and the goal is to maximize the total weight of covered elements. (It can be unweighted via standard duplication tricks.) For each $c \in C$, let $d_c := d(c, F')$ and let $F_c := \{f \in \mathcal{F} : d(f, c) < d_c\}$. Let $F_c = \{f_1, \dots, f_t\}$ ordered in the decreasing order of $d(f_i, c)$. We create t elements $E_c := \{e_{c,1}, \dots, e_{c,t}\}$ where $w(e_{c,1}) = d_c - d(f_1, c)$ and $w(e_{c,j}) = d(f_{j-1}, c) - d(f_j, c)$ for $j \in \{2, \dots, t\}$. We will have a set S_f for each $f \in \mathcal{F}$, and for each $c \in C$, if $f = f_i$ in F_c 's ordering, $S_f \cap E_c = \{e_{c,1}, \dots, e_{c,i}\}$. (If $f \notin F_c$, $S_f \cap E_c = \emptyset$.)

Then for any $F \subseteq \mathcal{F}$ and for any client $c \in C$, our construction ensures that $(\cup_{f \in F} S_f) \cap E_c$ is equal to $S_{f_c} \cap E_c$ where f_c is the closest facility to c in F , and the total weight of $(\cup_{f \in F} S_f) \cap E_c$ is exactly equal to $d(F', c) - d(f_c, c)$, which is exactly the improvement of the cost of c in $F \cup F'$ compared to F' . Since $\text{improv}(\cdot)$ is the sum of all clients, this function is a coverage function.

Proof of Theorem 11. The algorithm \mathcal{A} on input $((V, d), C, \mathcal{F}, \tau)$ using the notion of *coresets* and exhaustive enumerations (and using randomness), in FPT time constructs an instance for each guess of the values of ℓ_1, \dots, ℓ_k and R_1, \dots, R_k . The choice of ε in the use of coresets will be specified later. Thus, for a fixed instance (with ℓ_1, \dots, ℓ_k and R_1, \dots, R_k fixed), construct F_c for all $c \in C$, and then the (weighted) set-system $\{S_f\}_{f \in \mathcal{F}}$ over the universe $\cup_{c \in C} E_c$. Then, following the exact same calculations as in (2), we have that for a solution $S^* \subseteq \mathcal{F}$ such that $\text{improv}(S) \geq (1 - \frac{\alpha}{2}) \cdot \text{improv}(F^*)$, (for some $\alpha \geq 0$), we have:

$$\begin{aligned} \text{cost}(C, S^*) &= \text{cost}(C, F') - \text{improv}(S^*) \\ &\leq \text{cost}(C, F') - \left(1 - \frac{\alpha}{2}\right) \cdot \text{improv}(F^*) \\ &\leq \text{cost}(C, F') - \left(1 - \frac{\alpha}{2}\right) \cdot (\text{cost}(C, F') - \text{OPT}) \\ &= \left(1 - \frac{\alpha}{2}\right) \cdot \text{OPT} + \frac{\alpha}{2} \cdot \text{cost}(C, F') \\ &\leq \left(1 - \frac{\alpha}{2}\right) \cdot \text{OPT} + \frac{\alpha}{2} \cdot (3 + \varepsilon) \cdot \text{OPT} = \left(1 + \alpha + \frac{\alpha\varepsilon}{2}\right) \cdot \text{OPT}, \end{aligned}$$

where the last inequality follows from (1). The theorem statement completeness and soundness claims then follows by choosing $\varepsilon = \frac{2\delta}{\alpha}$. Moreover, the reduction is clearly in FPT time, and the weights of the elements we constructed are bounded by Δ , which is the blowup that happens in the size of the set system to reduce to the unweighted multicolored k -maxcoverage problem.

► **Remark 12.** The theorem statement also holds for the k -means objective as (1) is revised to $\text{cost}_2(C, F') \leq (9 + \varepsilon) \cdot \text{OPT}$ and we can thus conclude that for a solution $S^* \subseteq \mathcal{F}$ such that $\text{improv}(S) \geq (1 - \frac{\alpha}{8}) \cdot \text{improv}(F^*)$, we have $\text{cost}_2(C, S^*) \leq (1 - \frac{\alpha}{8}) \cdot \text{OPT} + \frac{\alpha}{8} \cdot (9 + \varepsilon) \cdot \text{OPT} = (1 + \alpha + \frac{\alpha\varepsilon}{8}) \cdot \text{OPT}$. ◀

► **Remark 13.** Starting from the multicolored k -maxcoverage problem, we can apply Theorem 5 to obtain a gap preserving reduction to 2-CSP, and then simply apply Lemma 19 in [14] to obtain a gap preserving reduction to (uncolored) k -maxcoverage problem. However, at the moment we do not know how to directly reduce the multicolored k -maxcoverage problem to the uncolored version (while retaining the gap), but this convoluted procedure suggests that a direct reduction might be plausible.

At this point one may wonder if it is possible to provide a gap preserving FPT reduction from the multicolored k -maxcoverage problem to the (unmulticolored) k -maxcoverage problem. Such a reduction would help us avoid Remark 10 and directly compose the results of

Theorems 11 and 5 in a blackbox manner. While reductions from the multicolored variant of a problem to its uncolored counterpart can be quite straightforward, such as for the k -clique and k -set cover problem, it can also be quite notorious such as for the k -set intersection problem [32, 6]. Such reductions are aptly dubbed as “reversing color coding”. To the best of our knowledge, reversing color coding for the k -maxcoverage problem can be quite hard (if we want to preserve some positive constant gap). That said, Remark 13 does provide a convoluted argument as to while a direct reversing color coding is currently out of reach, it can still be plausibly achieved. We note here that if we are promised that the multicolored k -maxcoverage problem instance has all the input sets of roughly the same size then there is a simple reduction to the (uncolored) k -maxcoverage problem by simply introducing k new subuniverses, one for each color class.

5 Inapproximability of k -MaxCoverage

In this section, we prove (the $W[1]$ -hardness part of) Theorem 2. In [31], the authors had *implicitly* proved the following: for some computable function F , it is $W[1]$ -hard given $(\mathcal{U}, \mathcal{S})$ to find k sets in \mathcal{S} that cover $1 - 1/F(k)$ fraction of \mathcal{U} whenever there exists k sets in \mathcal{S} that cover \mathcal{U} . We show below how that can be extended to obtain the more generalized result given in Theorem 2.

Our proof builds on the following $W[1]$ -hardness of gap k -MaxCover proved in [31].

► **Theorem 14** ([31]). *There exists a computable function $A: \mathbb{N} \rightarrow \mathbb{N}$ such that it is $W[1]$ -hard to decide an instance $\Gamma = (G = (V \dot{\cup} W, E), 1, \frac{1}{2})$ of k -MaxCover even in the following setting:*

- $V := V_1 \dot{\cup} \dots \dot{\cup} V_k$, where $\forall j \in [k], |V_j| = n$.
- $W := W_1 \dot{\cup} \dots \dot{\cup} W_\ell$, where $\ell = (\log n)^{O(1)}$ and $\forall i \in [k], |W_i| = A(k)$.

Proof Sketch. All the references here are using the labels in [31]. First we apply Proposition 5.1 to Theorem 7.1 with $z = \frac{1}{\log_2(\frac{1}{1-\delta})}$ to obtain a $(0, O(\log_2 m), O(t), 1/2)$ -efficient protocol for k -player $\text{MultEq}_{m,k,t}$ in the SMP model. The proof of the theorem then follows by plugging in the parameters of the protocol to Corollary 5.5. ◀

Starting from the above theorem, we mimics ideas from Feige’s proof of the NP-hardness of approximating the Max Coverage problem [20]. Let $T := T(k)$ be an integer such that $\rho(Tk) \geq 2 \cdot k^{A(k)}$ (such an integer T exists because ρ is unbounded). Given an instance $\Gamma(G = (V \dot{\cup} W, E), 1, 1/2)$ of k -MaxCover, we construct the universe $\mathcal{U} := \{(t, i, f) : t \in [T], i \in [\ell], f: W_i \rightarrow [k]\}$, and a collection of sets $\mathcal{S} := \{S_{(t,j,v)}\}_{t \in [T], j \in [k], v \in V_j}$, where for all $t \in [T]$ we have $(t, i, f) \in S_{(t,j,v)} \iff \exists w \in W_i$ such that $f(w) = j$ and $(v, w) \in E$. Note that $|\mathcal{U}| = T \cdot (\log n)^{O(1)} \cdot k^{A(k)}$ and that $|\mathcal{S}| = T \cdot k \cdot n$.

Suppose there exists $v_1 \in V_1, \dots, v_k \in V_k$ such that for all $i \in [\ell]$ we have that W_i is covered by v_1, \dots, v_k . Let $w_i \in W_i$ be a common neighbor of v_1, \dots, v_k . Then, we claim that the collection $\{S_{(t,1,v_1)}, S_{(t,2,v_2)}, \dots, S_{(t,k,v_k)}\}_{t \in [T]}$ covers \mathcal{U} . This is because for every $t \in [T]$ and every $(t, i, f) \in \mathcal{U}$ we have that $S_{(t,f(w_i),v_{f(w_i)})}$ covers it.

On the other hand, suppose that for every $v_1 \in V_1, \dots, v_k \in V_k$ we have that only $1/2$ fraction of the W_i s are covered by v_1, \dots, v_k . Fix some Tk sets $\tilde{\mathcal{S}}$ in \mathcal{S} . For every $t \in [T]$, let $\mathcal{U}_t := \{(t, i, f) : i \in [\ell], f: W_i \rightarrow [k]\}$ and $\tilde{\mathcal{S}}_t := \tilde{\mathcal{S}} \cap \{S_{(t,j,v)} : j \in [k], v \in V_j\}$. We can partition $\tilde{\mathcal{S}}$ to $\tilde{\mathcal{S}}^-, \tilde{\mathcal{S}}^=$, and $\tilde{\mathcal{S}}^+$, where for every $t \in [T]$ we include all the sets in $\tilde{\mathcal{S}}_t$ to $\tilde{\mathcal{S}}^-$ if $|\tilde{\mathcal{S}}_t| < k$, to $\tilde{\mathcal{S}}^+$ if $|\tilde{\mathcal{S}}_t| > k$, and to $\tilde{\mathcal{S}}^=$ if $|\tilde{\mathcal{S}}_t| = k$. Let $R^+ \subseteq [T]$ (resp. $R^- \subseteq [T]$) be defined as follows: $t \in R^+ \iff |\tilde{\mathcal{S}}_t| > k$ (resp. $t \in R^- \iff |\tilde{\mathcal{S}}_t| < k$). Since $|\tilde{\mathcal{S}}| = Tk$, we have that there exists $Q \subseteq [T] \times [k]$ such that $|Q| = |\tilde{\mathcal{S}}^+| - (|R^+| \cdot k)$ and $(t, j) \in Q \iff \tilde{\mathcal{S}}_t \cap \{S_{(t,j,v)} : v \in V_j\} = \emptyset$ and $|\tilde{\mathcal{S}}_t| < k$.

Now, we observe that once we fix $t \in [T]$ and $j \in [k]$ and if $\tilde{\mathcal{S}}_t \cap \{S_{(t,j,v)} : v \in V_j\} = \emptyset$ then $\tilde{\mathcal{S}}$ does not cover any element in the subuniverse $\{(t, i, f_j) : i \in [\ell]\}$, where f_j is the constant function which maps everything to j . Therefore, we can conclude that there are $|Q| \cdot \ell$ many elements in $\bigcup_{t \in R^-} \mathcal{U}_t$ such that are not covered by $\tilde{\mathcal{S}}$.

Also, suppose we picked $S_{(t,1,v_1)}, S_{(t,2,v_2)}, \dots, S_{(t,k,v_k)}$ for some $t \in [T]$, $v_1 \in V_1, \dots, v_k \in V_k$ then for every $i \in [\ell]$ such that v_1, \dots, v_k does not cover W_i , we have that for every $w \in W_i$ there is some $j \in [k]$ such that $(v_j, w) \notin E$. Therefore, there is some $f : W_i \rightarrow [k]$ such that $S_{(t,1,v_1)}, S_{(t,2,v_2)}, \dots, S_{(t,k,v_k)}$ does not cover (t, i, f) . Thus, these k sets do not cover at least $\ell/2$ universe elements.

We can now put everything together to complete the soundness analysis. First note that for every $t \in [T]$, no element of the subuniverse \mathcal{U}_t can be covered by a set in $\tilde{\mathcal{S}} \setminus \tilde{\mathcal{S}}_t$. For every $t \in [T]$, if $|\tilde{\mathcal{S}}_t| = k$ then from the above analysis we have that at least $\ell/2$ elements of \mathcal{U}_t are not covered by $\tilde{\mathcal{S}}$.

Therefore, the total number of universe elements not covered by $\tilde{\mathcal{S}}$ is at least $(|\tilde{\mathcal{S}}^-| \cdot \ell/2k) + |Q| \cdot \ell = \ell \cdot (|\tilde{\mathcal{S}}^-|/2k + |\tilde{\mathcal{S}}^+| - (|R^+| \cdot k))$. Since $|\tilde{\mathcal{S}}^-| = (T - R^+ - R^-) \cdot k$, this implies that $\tilde{\mathcal{S}}$ does not cover at least $\ell \cdot (T/2 + |\tilde{\mathcal{S}}^+| - (|R^+| \cdot k) - \frac{R^+ + R^-}{2})$ elements in \mathcal{U} . Next, we note that by the way we partitioned $\tilde{\mathcal{S}}$, we have $|\tilde{\mathcal{S}}^+| - (|R^+| \cdot k) = (|R^-| \cdot k) - |\tilde{\mathcal{S}}^-|$, and we also have $|\tilde{\mathcal{S}}^+| \geq |R^+| \cdot (k + 1)$ and $|\tilde{\mathcal{S}}^-| \leq |R^-| \cdot (k - 1)$. From this we can surmise that $|\tilde{\mathcal{S}}^+| - (|R^+| \cdot k) \geq \max(|R^+|, |R^-|) \geq \frac{R^+ + R^-}{2}$. Thus, we have that $\tilde{\mathcal{S}}$ does not cover at least $\ell T/2$ universe elements.

Thus, we can conclude that $\tilde{\mathcal{S}}$ can not cover at least $T\ell/2$ universe elements. This is $k^{-A(k)}/2$ fraction of \mathcal{U} that is not covered by $\tilde{\mathcal{S}}$. The proof follows by noting that $k^{-A(k)}/2 \geq 1/\rho(Tk) = 1/\rho(|\tilde{\mathcal{S}}|)$.

References

- 1 Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998. doi:10.1145/278298.278306.
- 2 Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45(1):70–122, 1998. doi:10.1145/273865.273901.
- 3 Huck Bennett, Mahdi Cheraghchi, Venkatesan Guruswami, and João Ribeiro. Parameterized inapproximability of the minimum distance problem over all fields and the shortest vector problem in all lp norms. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 553–566. ACM, 2023. doi:10.1145/3564246.3585214.
- 4 Arnab Bhattacharyya, Édouard Bonnet, László Egri, Suprovat Ghoshal, Karthik C. S., Bingkai Lin, Pasin Manurangsi, and Dániel Marx. Parameterized intractability of even set and shortest vector problem. *J. ACM*, 68(3):16:1–16:40, 2021. doi:10.1145/3444942.
- 5 Arnab Bhattacharyya, Suprovat Ghoshal, Karthik C. S., and Pasin Manurangsi. Parameterized intractability of even set and shortest vector problem from gap-eth. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 17:1–17:15, 2018. doi:10.4230/LIPIcs.ICALP.2018.17.
- 6 Boris Bukh, Karthik C. S., and Bhargav Narayanan. Applications of random algebraic constructions to hardness of approximation. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 237–244. IEEE, 2021. doi:10.1109/FOCS52979.2021.00032.
- 7 Grigori Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM Journal on Computing*, 40(6):1740–1766, 2011. doi:10.1137/080733991.

- 8 Parinya Chalermsook, Marek Cygan, Guy Kortsarz, Bundit Laekhanukit, Pasin Manurangsi, Danupon Nanongkai, and Luca Trevisan. From gap-exponential time hypothesis to fixed parameter tractable inapproximability: Clique, dominating set, and more. *SIAM J. Comput.*, 49(4):772–810, 2020. doi:10.1137/18M1166869.
- 9 Yijia Chen, Yi Feng, Bundit Laekhanukit, and Yanlin Liu. Simple combinatorial construction of the $k^{O(1)}$ -lower bound for approximating the parameterized k -clique. *CoRR*, abs/2304.07516, 2023. doi:10.48550/arXiv.2304.07516.
- 10 Yijia Chen and Bingkai Lin. The constant inapproximability of the parameterized dominating set problem. *SIAM J. Comput.*, 48(2):513–533, 2019. doi:10.1137/17M1127211.
- 11 Rajesh Chitnis, Andreas Emil Feldmann, and Pasin Manurangsi. Parameterized approximation algorithms for bidirected steiner network problems. *ACM Transactions on Algorithms (TALG)*, 17(2):1–68, 2021. doi:10.1145/3447584.
- 12 Rajesh Chitnis, Andreas Emil Feldmann, and Ondrej Suchý. A tight lower bound for planar steiner orientation. *Algorithmica*, 81(8):3200–3216, 2019. doi:10.1007/s00453-019-00580-x.
- 13 Vincent Cohen-Addad, Anupam Gupta, Amit Kumar, Euiwoong Lee, and Jason Li. Tight fpt approximations for k -median and k -means. *arXiv preprint*, 2019. arXiv:1904.12334.
- 14 Vincent Cohen-Addad, Anupam Gupta, Amit Kumar, Euiwoong Lee, and Jason Li. Tight FPT approximations for k -median and k -means. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPICs*, pages 42:1–42:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.ICALP.2019.42.
- 15 Vincent Cohen-Addad Viallat, Fabrizio Grandoni, Euiwoong Lee, and Chris Schwiegelshohn. Breaching the 2 lmp approximation barrier for facility location with applications to k -median. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 940–986. SIAM, 2023.
- 16 Pierluigi Crescenzi, Riccardo Silvestri, and Luca Trevisan. On weighted vs unweighted versions of combinatorial optimization problems. *Inf. Comput.*, 167(1):10–26, 2001. doi:10.1006/inco.2000.3011.
- 17 Irit Dinur. The PCP theorem by gap amplification. *J. ACM*, 54(3):12, 2007. doi:10.1145/1236457.1236459.
- 18 Irit Dinur. Mildly exponential reduction from gap 3sat to polynomial-gap label-cover. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:128, 2016. URL: <http://eccc.hpi-web.de/report/2016/128>, arXiv:TR16-128.
- 19 Pavel Dvořák, Andreas Emil Feldmann, Ashutosh Rai, and Paweł Rzażewski. Parameterized inapproximability of independent set in h -free graphs. *Algorithmica*, pages 1–27, 2022.
- 20 Uriel Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998. doi:10.1145/285055.285059.
- 21 Uriel Feige, Shafi Goldwasser, László Lovász, Shmuel Safra, and Mario Szegedy. Interactive proofs and the hardness of approximating cliques. *J. ACM*, 43(2):268–292, 1996. doi:10.1145/226643.226652.
- 22 Andreas Emil Feldmann, Karthik C. S., Euiwoong Lee, and Pasin Manurangsi. A survey on approximation in parameterized complexity: Hardness and algorithms. *Algorithms*, 13(6):146, 2020. doi:10.3390/a13060146.
- 23 Sudipto Guha and Samir Khuller. Greedy strikes back: Improved facility location algorithms. *Journal of algorithms*, 31(1):228–248, 1999. doi:10.1006/JAGM.1998.0993.
- 24 Venkatesan Guruswami, Bingkai Lin, Xuandi Ren, Yican Sun, and Kewen Wu. Almost optimal time lower bound for approximating parameterized clique, csp, and more, under ETH. *CoRR*, abs/2404.08870, 2024. doi:10.48550/arXiv.2404.08870.
- 25 Venkatesan Guruswami, Bingkai Lin, Xuandi Ren, Yican Sun, and Kewen Wu. Parameterized inapproximability hypothesis under ETH. In *STOC*, 2024.

- 26 Dorit S Hochba. Approximation algorithms for np-hard problems. *ACM Sigact News*, 28(2):40–52, 1997. doi:10.1145/261342.571216.
- 27 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.
- 28 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.
- 29 Tapas Kanungo, David M Mount, Nathan S Netanyahu, Christine D Piatko, Ruth Silverman, and Angela Y Wu. A local search approximation algorithm for k-means clustering. *Computational Geometry*, 28(2-3):89–112, 2004. doi:10.1016/J.COMGEO.2004.03.003.
- 30 Karthik C. S. and Subhash Khot. Almost polynomial factor inapproximability for parameterized k-clique. In Shachar Lovett, editor, *37th Computational Complexity Conference, CCC 2022, July 20-23, 2022, Philadelphia, PA, USA*, volume 234 of *LIPICs*, pages 6:1–6:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.CCC.2022.6.
- 31 Karthik C. S., Bundit Laekhanukit, and Pasin Manurangsi. On the parameterized complexity of approximating dominating set. *J. ACM*, 66(5):33:1–33:38, 2019. doi:10.1145/3325116.
- 32 Karthik C. S. and Pasin Manurangsi. On closest pair in euclidean metric: Monochromatic is as hard as bichromatic. *Combinatorica*, 40(4):539–573, 2020. doi:10.1007/S00493-019-4113-1.
- 33 Karthik C. S. and Inbal Livni Navon. On hardness of approximation of parameterized set cover and label cover: Threshold graphs from error correcting codes. In Hung Viet Le and Valerie King, editors, *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*, pages 210–223. SIAM, 2021. doi:10.1137/1.9781611976496.24.
- 34 Subhash Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the thirty-fourth annual ACM symposium on Theory of computing*, pages 767–775. ACM, 2002. doi:10.1145/509907.510017.
- 35 Subhash Khot, Dor Minzer, and Muli Safra. On independent sets, 2-to-2 games, and grassmann graphs. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 576–589. ACM, 2017. doi:10.1145/3055399.3055432.
- 36 Subhash Khot, Dor Minzer, and Muli Safra. Pseudorandom sets in grassmann graph have near-perfect expansion. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 592–601. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00062.
- 37 Vladimir Kolmogorov, Andrei A. Krokhin, and Michal Rolínek. The complexity of general-valued csp. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 1246–1258. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.80.
- 38 Bingkai Lin. The parameterized complexity of the k-biclique problem. *J. ACM*, 65(5):34:1–34:23, 2018. doi:10.1145/3212622.
- 39 Bingkai Lin. A simple gap-producing reduction for the parameterized set cover problem. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, pages 81:1–81:15, 2019. doi:10.4230/LIPICs.ICALP.2019.81.
- 40 Bingkai Lin. Constant approximating k-clique is w [1]-hard. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1749–1756, 2021. doi:10.1145/3406325.3451016.
- 41 Bingkai Lin, Xuandi Ren, Yican Sun, and Xiuhan Wang. Improved hardness of approximating k-clique under ETH. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 285–306. IEEE, 2023. doi:10.1109/FOCS57990.2023.00025.
- 42 Bingkai Lin, Xuandi Ren, Yican Sun, and Xiuhan Wang. Constant approximating parameterized k-setcover is W[2]-hard. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 3305–3316. SIAM, 2023. doi:10.1137/1.9781611977554.CH126.

- 43 Daniel Lokshantov, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Parameterized complexity and approximability of directed odd cycle transversal. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2181–2200, 2020. doi:10.1137/1.9781611975994.134.
- 44 Pasin Manurangsi. Tight running time lower bounds for strong inapproximability of maximum k -coverage, unique set cover and related problems (via t -wise agreement testing theorem). In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 62–81. SIAM, 2020. doi:10.1137/1.9781611975994.5.
- 45 Pasin Manurangsi and Prasad Raghavendra. A birthday repetition theorem and complexity of approximating dense csps. *CoRR*, abs/1607.02986, 2016. arXiv:1607.02986.
- 46 Naoto Ohsaka. On the parameterized intractability of determinant maximization. *arXiv preprint*, 2022. doi:10.48550/arXiv.2209.12519.
- 47 Michal Włodarczyk. Inapproximability within $W[1]$: the case of steiner orientation. *CoRR*, abs/1907.06529, 2019. arXiv:1907.06529.



On Controlling Knockout Tournaments Without Perfect Information

Václav Blažej  

University of Warwick, Coventry, UK

Sushmita Gupta  

The Institute of Mathematical Sciences, HBNI, Chennai, India

M. S. Ramanujan  

University of Warwick, Coventry, UK

Peter Strulo  

University of Warwick, Coventry, UK

Abstract

Over the last decade, extensive research has been conducted on the algorithmic aspects of designing single-elimination (SE) tournaments. Addressing natural questions of algorithmic tractability, we identify key properties of input instances that enable the tournament designer to efficiently schedule the tournament in a way that maximizes the chances of a preferred player winning. Much of the prior algorithmic work on this topic focuses on the perfect (complete and deterministic) information scenario, especially in the context of fixed-parameter algorithm design. Our contributions constitute the first fixed-parameter tractability results applicable to more general settings of SE tournament design with potential imperfect information.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Parameterized algorithms, Tournament design, Imperfect information

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.7

Funding Supported by the Engineering and Physical Sciences Research Council (grant number EP/V044621/1).

Sushmita Gupta: Supported by SERB's MATRICS Grant (MTR/2021/000869) and SUPRA Grant (SPR/2021/000860).

1 Introduction

The algorithmic aspects of designing single-elimination (SE) knockout tournaments has been the subject of extensive study in the last decade. This format of competition, prevalent in various scenarios like sports, elections, and decision-making processes [28, 14, 24, 20, 7], typically involves multiple rounds, ultimately leading to a single winner. Assume that the number of players n is a power of 2 and consider a complete binary tree T of depth $\log n$. We can assign each player to a leaf of T , which gives us an initial set of pairings where the player at a leaf is paired with the player at the sibling of the leaf. In the first round, the players in each pair play against each other. Then, each loser is knocked out, we assign each winner to its parent node in T , and have the new siblings play against each other. Eventually there will only be one player remaining and they will be assigned to the root of the tree: they are declared the winner. That is, in round i , we label the non-leaf nodes at height $i - 1$ (leaves have height 0) with the winner of the match between the labels of its children. Finally, the label assigned to the root node of T is the overall winner of the SE tournament.



© Václav Blažej, Sushmita Gupta, M. S. Ramanujan, and Peter Strulo;
licensed under Creative Commons License CC-BY 4.0

19th International Symposium on Parameterized and Exact Computation (IPEC 2024).

Editors: Édouard Bonnet and Paweł Rzażewski; Article No. 7; pp. 7:1–7:15

Leibniz International Proceedings in Informatics



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

7:2 On Controlling Knockout Tournaments Without Perfect Information

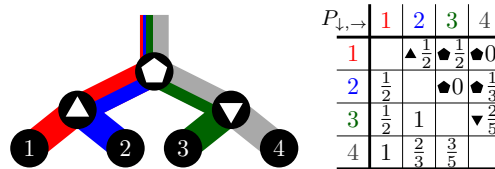
A major research direction in this topic revolves around the algorithmic efficiency of computing the initial pairings in a way that maximizes the chances of a chosen player winning the resulting SE tournament. Here, one aims to understand the theory behind dynamics and vulnerabilities of such tournament designs from an algorithmic perspective.

When the designer is told the winner of a match between players i and j for every pair of players i, j (i.e., the information is “complete”) and these predictions are certain (i.e., the information is deterministic), we get a model with perfect information and in this case, the problem is termed Tournament Fixing (TF). The goal of the designer is then to decide whether there is a mapping of the players to the leaves of the complete binary tree T (this mapping is called a seeding) that results in the chosen player winning the resulting SE tournament. This special case of complete and deterministic information has already garnered considerable attention in the literature, particularly from the perspective of parameterized complexity starting with the work of Ramanujan and Szeider [23] followed by [9, 10, 11, 12, 5] who parameterize with *feedback arc set* number, and [31] with *feedback vertex set* number.

A significant gap in the extensive literature that has been identified in several papers (see, for example [23, 10, 31, 5]), is that of the parameterized complexity of SE tournament design problems in cases where the designer *does not* have perfect information. In this paper, we aim to bridge this gap by presenting a novel parameterized algorithm for a problem we call PROBABILISTIC TOURNAMENT FIXING (PTF). The input to PTF consists of an $n \times n$ matrix where (i) the (i, j) ’th entry is denoted by $P_{i,j}$ and is some value in $[0, 1]$, and (ii) $P_{i,j} = 1 - P_{j,i}$, see Figure 1. The goal is to decide if there is a seeding such that the probability of the chosen player, α^* , winning the resulting tournament is at least a given value, p^* .

As well as more closely modeling the information available in real tournaments, PTF is a generalization of TF that has applications to other perspectives. For example, tournament fixing can be seen as a special case of agenda control [2, 3]—a type of manipulation where a centralized authority tries to enforce an outcome. In the backdrop of probabilistic tournament fixing, one can ask the analogous control question: Can the tournament designer ensure that a chosen player wins the tournament even when it does not have complete information about the outcome of all possible matches? This “robust” design objective can be expressed as an instance of PTF where the target probability $p^* = 1$. Any uncertain matches can be assigned any non-integer probability: if the winner of the tournament depends on them then α^* will not win with probability 1. This also addresses the scenario where some matches are susceptible to collusion between the players and so the designer wants to guarantee the result regardless of this adversarial behavior.

All of the aforementioned papers on tournament design problems that use parameterized algorithms are in the *deterministic* setting; and to the best of our knowledge, this is the first work on probabilistic tournament fixing from the perspective of parameterized complexity.



■ **Figure 1** An example of PTF. Thickness represents probability of a player winning and probabilities that influence a match are marked by shapes. Here, player 4 wins with probability 6/10.

Our contributions

To obtain our parameterized algorithm for PTF we introduce and solve a new problem that we call SIMULTANEOUS TF (STF). Unlike PTF, STF is completely deterministic. As it is easier to state our results for STF in terms of input digraphs rather than matrices, let us first introduce the following notation.

A *certainty digraph* is a graphical representation of the integral values in a given set of pairwise winning probabilities. It is a directed graph with the set of players as vertices, where the arc ij exists if and only if i deterministically beats j (i.e., $P_{i,j} = 1$ and $P_{j,i} = 0$). A *tournament digraph* is simply the certainty digraph of a matrix of pairwise winning probabilities where every possible value is present and it is either 0 or 1. Note that in a tournament digraph, there is exactly one arc between every pair of vertices, but this is not necessarily true of a certainty digraph in general.

In STF, one is given a sequence of tournament digraphs D_1, \dots, D_m over the same set of n players and a chosen player α^* . The goal in STF is to compute a *single seeding* (if one exists) that makes α^* win in the SE tournament resulting from D_i for *every* i . It is straightforward to see that this problem is NP-hard. For $m = 1$, it is just TF. We obtain a novel fixed-parameter (FPT) algorithm for STF parameterized by two parameters, k_1 and k_2 where (i) the m tournament digraphs agree on the orientations of all but at most k_1 arcs, and (ii) the largest digraph that appears as a common subgraph of every D_i , has a feedback arc set number (see Section 2 for a formal definition) of at most k_2 . That is, we obtain an algorithm with running time $f(k_1, k_2)n^{\mathcal{O}(1)}$ for some function f . So, we get polynomial-time solvability when both k_1 and k_2 are bounded. We also show that dropping either parameter leads to Para-NP-hardness. That is, unless $P=NP$, there is no FPT algorithm for STF parameterized by k_1 alone or by k_2 alone.

Implications for PTF. We obtain an FPT algorithm for PTF parameterized by c_1 and c_2 , where c_1 is the number of vertex pairs $\{i, j\}$ for which the given value of $P_{i,j}$ is non-integral, and c_2 is the feedback arc set number of the certainty digraph of the given set of pairwise winning probabilities. As a consequence, we get polynomial-time solvability of PTF as long as the number of fractional entries in the input is bounded and a natural digraph defined by the integral entries has bounded feedback arc set number.

Implications for TF. Our FPT algorithm for STF generalizes and significantly extends the known fixed-parameter tractability of TF parameterized by the feedback arc set number of the input tournament digraph [23, 9, 11]. Indeed, if we set $m = 1$, implying that $k_1 = 0$, the value of k_2 is precisely this parameter.

Our motivation behind the choice of parameter

Small feedback arc set number (FAS) is a natural condition for competitions where there is a clear-cut ranking of players with a small number of possible upsets. The relevance of this parameter is evidenced by empirical work [25] and has also received significant attention in the theoretical literature on this family of problems [1, 23, 9, 11, 12]. Our combined parameter for PTF is in some sense a generalization of this well-motivated parameter: arcs corresponding to fractional entries (c_1) can have either orientation and hence can contribute to the eventual FAS, while feedback arcs in the certainty digraph are already counted by c_2 . Note that [31] gives an FPT algorithm parameterized by feedback *vertex* set number but this does not subsume our work since it is restricted to the deterministic setting.

Related work

The influential work of Vu et al. [29] has inspired a long line of work in the topic of tournament fixing [30, 27, 26, 17, 23, 10, 9, 11, 31, 5, 12] primarily in the deterministic setting. The probabilistic setting, which captures imperfect predictions of games, has also received some attention over the years, [30, 27, 26, 1]. However, the results in these papers are not algorithmic and do not necessarily hold for any given instance of PTF. Specifically, [30, 27, 26] study the Braverman-Mossel probabilistic model for tournament generation and the existence of desirable properties such as back matchings, seedings where more than half of the top-players can win, and so on.

The topic of agenda control, initiated by Bartholdi et al. [2, 3], within which the problems on tournament fixing lie, is of significant interest in computational social choice and algorithmic game theory. Tournament fixing as a form of agenda control has been discussed formally in [4, Chapter 19].

Single-elimination tournaments have strong ties to a specific category of elections extensively explored in voting theory, namely sequential elimination voting with pairwise comparison. This is a vast area of research and we mention a few works that are most closely related to our setting. Hazon et al. [13] study the algorithmic aspects of rigging elections based on the shape of the voting tree and assuming probabilistic information. Additionally, Mattei et al. [21] study the complexity of bribery and manipulation problems in sports tournaments with probabilistic information. Recently, [6] has studied the parameterized complexity of some of these bribery questions focusing mainly on another variety of tournaments but also giving an intractability result on SE tournaments. Furthermore, Konczak et al. [18], Lang et al. [19] and Pini et al. [22], study sequential elimination voting with incompletely specified preferences. Their objectives include the identification of winning candidates in either some or all complete extensions of partial preference profiles based on a given voting rule. This line of work is related to the special case of PTF where the target probability is 1.

2 Preliminaries

We work only with directed graphs $G = (V(G), E(G))$ and refer to directed edges (i.e., arcs) as $uv \in E(G)$; note $uv \neq vu$. We use the notation $[n] = \{1, \dots, n\}$ and use $[a, b]$ to represent values from a range between a and b . We make the standard assumption of dealing with inputs comprising rational numbers (see, for example, [21]). Throughout the following we will fix the number of players as n and T as the perfect binary tree with n leaves. We denote the leaves by $L(T)$. We will assume that n is a power of 2 and that the number of levels of T is $\log n$ (an integer). We use $\text{Desc}(v)$ to refer to the descendants of v in T (including v), i.e., the vertices w such that there is a path from v to w away from the root. Similarly the ancestors of v are the vertices w with a path from v to w towards the root. We define $\text{height}(v)$ as the number of edges on a path between v and its closest leaf.

A Q -seeding is a function $\gamma: L(T) \rightarrow Q$. If we choose Q as our set of players and our seeding is bijective, this definition coincides with Definition 1 from [29] except that we require that T is always the perfect binary tree with n leaves, whereas they accept any binary tree.

► **Definition 1 (Brackets).** Given tournament digraph D and a $V(D)$ -seeding γ , a *bracket generated by γ with respect to D* is a labeling of T , defined as $\ell: V(T) \rightarrow V(D)$ such that $\ell(v) = \gamma(v)$ for every leaf $v \in L$ and for every inner node v of T with children u and w , $\ell(v) = \ell(u)$ if $\ell(u)\ell(w) \in E(D)$ and $\ell(v) = \ell(w)$ otherwise. The player that labels the root of T is said to *win* the bracket ℓ .

► **Definition 2.** A *feedback arc set* (FAS) of a digraph D is a set of arcs, called *back arcs*, whose reversal makes the digraph acyclic. The *FAS number* of D is the size of a smallest FAS.

Note that for the graph with reversed back arcs we can devise a topological vertex ordering \prec such that the set of back arcs in the original graph D is $\{xy \in E(D) \mid y \prec x\}$. Note that a tournament digraph is acyclic if and only if it is also transitive.

In proofs, we use the following technical folklore tool.

► **Lemma 3.** For every $k, n \in \mathbb{N}$ where $k \leq n$, we have $(\log n)^k \leq (4k \log k)^k + n^2$

We will use as a subroutine the well-known FPT algorithm for ILP-FEASIBILITY. The ILP-FEASIBILITY problem is defined as follows. The input is a matrix $A \in \mathbb{Z}^{m \times p}$ and a vector $b \in \mathbb{Z}^{m \times 1}$ and the objective is to find a vector $\bar{x} \in \mathbb{Z}^{p \times 1}$ satisfying the m inequalities given by A , that is, $A \cdot \bar{x} \leq b$, or decide that such a vector does not exist.

► **Proposition 4** ([15, 16, 8]). ILP-FEASIBILITY can be solved using $\mathcal{O}(p^{2.5p+o(p)} \cdot L)$ arithmetic operations and space polynomial in L , where L is the number of bits in the input and p is the number of variables.

3 The Algorithm for STF and its Analysis

In this section we will present the algorithm for STF. In Section 4 we will show the application to PTF. We remark that a reader interested in the application can skip ahead to this section immediately after parsing the main statement (Theorem 5).

Recall that the STF problem is formally defined as follows.

SIMULTANEOUS TOURNAMENT FIXING (STF)

Input: A sequence of tournament digraphs D_1, \dots, D_m on vertex set N and a player $\alpha^* \in N$.

Question: Does there exist an N -seeding γ such that for all $i \in [m]$, α^* wins the bracket generated by γ with respect to D_i ?

Given tournament digraphs D_1, \dots, D_m on common vertex set N , we define the set of *shared arcs*, $\widehat{E} = \bigcap_{i=1}^m E(D_i)$. The remaining vertex pairs are the *private arcs*. The *shared digraph* is the graph (N, \widehat{E}) . We call the FAS of the shared digraph the *shared FAS* and denote it \widehat{F} and let \prec denote the ordering where the back arcs of the shared FAS is the set $\{xy \in \widehat{E} \mid y \prec x\}$. This ordering is typically depicted left-to-right so for $y \prec x$ we also say y is *left of* x and x is *right of* y .

► **Theorem 5.** STF is FPT parameterized by the size of the shared FAS and the number of private arcs.

We argue that *both* parameters in the above statement are required, by showing that STF is NP-hard even when either one of the parameters is a constant (i.e., STF is para-NP-hard parameterized by either parameter alone).

► **Lemma 6.** STF parameterized by the shared FAS is para-NP-hard and STF parameterized by the number of private arcs is para-NP-hard.

Proof. Notice that if $m = 1$, then the number of private arcs is 0 and we get the TF problem, which is NP-hard. On the other hand, let (D, α^*) be an instance of TF. Construct an instance (D_1, D_2, α^*) of STF by setting $D_1 = D$ and defining D_2 as the tournament obtained

7:6 On Controlling Knockout Tournaments Without Perfect Information

by taking an arbitrary acyclic (re-)orientation of the arcs of D , such that all arcs incident on α^* are oriented away from it (i.e., α^* beats everyone else). Then, it is easy to see that the constructed instance of STF is a yes-instance if and only if the original TF instance is also a yes-instance. Moreover, the shared digraph in the STF instance is a subgraph of D_2 and so is acyclic, i.e., the instance has a shared FAS of size 0. ◀

The rest of the section is devoted to the proof of Theorem 5. For ease of description, in the rest of this section we will denote our combined parameter (i.e., the sum of the size of the shared FAS and number of private arcs) by k . Note that we can assume that the given tournament digraphs are distinct and that $m \leq 2^k$. This holds because k upper bounds the number of private arcs and there are two possibilities of how a private arc may be present in a tournament digraph (either uv or vu) so having more than 2^k necessarily produces duplicate tournament digraphs.

Our parameterization allows us to define a small number of “interesting” vertices as follows. We say a vertex is *affected* if it is an endpoint of either a feedback arc or a private arc. Additionally, α^* is also affected. More precisely, the set of affected vertices is $V(\widehat{F}) \cup V(E(D_1) \setminus \widehat{E}) \cup \{\alpha^*\}$ denoted by $V_A = \{a_1, \dots, a_k\}$ where the ordering agrees with \prec (i.e. $a_j \prec a_{j+1}$). Note that $|V_A| \leq 2k + 1$.

We now use these affected vertices as “breakpoints” and classify the remaining vertices by where in the order they fall relative to the affected vertices. Define a set $\mathbf{Types} = [|V_A| + 1] \cup V_A$ and a *type function* $\tau: N \rightarrow \mathbf{Types}$ where $\tau(v) = v$ for each $v \in V_A$ and otherwise $\tau(v) = j$, where j is the smallest index in $[|V_A|]$ such that $v \prec a_j$. If there is no such index set $\tau(v) = |V_A| + 1$. We say a vertex v is of type t if $\tau(v) = t$. We refer to types in $\mathbf{Flex} = \mathbf{Types} \setminus V_A$ as *flexible* types and others as *singular* types. See Figure 2 for an example.

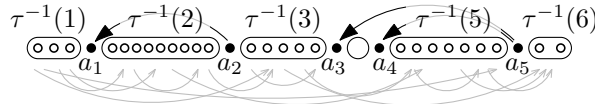
This “classification” of players is motivated by the following observation, which implies that for any three distinct players, two of which are of the same type, and the third is of a different type, the first two have the exact same win/loss relationship with the third one.

► **Observation 7** (slightly extended Lemma 2 in [23]). *For every $i \in [m]$, type $t \in \mathbf{Types}$, distinct players $u, v \in \tau^{-1}(t)$, and $w \in N$ such that $\tau(w) \neq t$, $wu \in E(D_i)$ if and only if $wv \in E(D_i)$.*

The above observation allows us to construct a tournament digraph on \mathbf{Types} . Moreover, in our search for a seeding which makes α^* win, Observation 8 implies that we can ignore the specific placement of the players that have the same type in relation to each other.

► **Observation 8.** *If two seedings γ_1 and γ_2 of the same tournament digraph D differ only in placement of players that have the same type, i.e., $\gamma_1(u) \neq \gamma_2(u) \implies \tau(\gamma_1(u)) = \tau(\gamma_2(u))$, then α^* wins the bracket generated by γ_1 with respect to D if and only if α^* wins the bracket generated by γ_2 with respect to D .*

This allows us to view the solution to STF as a seeding on the set \mathbf{Types} and fill in the actual players at a later stage. This insight leads us to the following definitions.



■ **Figure 2** The set $\mathbf{Types} = (1, a_1, 2, a_2, 3, a_3, 4, a_4, 5, a_5, 6)$ sorted according to \prec . The flexible types are depicted containing the players of that type; note $\tau^{-1}(4) = \emptyset$. The back arcs are depicted above while the remaining arcs go “right”, below the vertices we depict some of the remaining arcs for illustration.

Digraph, seeding and bracket of Types

We construct a tournament digraph, denoted by D_i^T , with vertex set **Types** where type x beats type y in D_i^T when vertices of type x beat vertices of type y in D_i . Precisely stated, $D_i^T = (\text{Types}, E)$ where $xy \in E$ if and only if there exist $u, v \in N$ such that $\tau(u) = x$, $\tau(v) = y$, and $uv \in E(D_i)$. We call D_i^T the *Types-digraph generated by D_i* . Since D_i^T is a tournament digraph we can define a topological ordering \prec in much the same way as for the original digraph. Since the flexible types are not affected vertices there are no back arcs so we have the following observation.

► **Observation 9.** *For all $x, y \in \text{Flex}$, $x \prec y$ if and only if $xy \in E(D_i^T)$ (i.e. x beats y).*

As discussed earlier, from now on we will use a **Types-seeding** to represent the solution and fill in the actual players in place of their types at the very end. Given an N -seeding γ , we define the **Types-seeding** $\beta: L \rightarrow \text{Types}$ as $\beta(u) = \tau(\gamma(u))$, for each $u \in L(T)$. This corresponds to taking the seeded players and mapping them to their types. Finally, for each $i \in [m]$, we define ℓ_i as the bracket generated by β with respect to D_i^T .

To exploit Observation 8 we focus on bracket vertices that may be influenced by differences between the input tournament digraphs. Bracket vertices that depend only on the shared digraph always have the same winner (Observation 11), but bracket vertices that have affected vertices as their descendants may have different winners in different input tournaments. This distinction inspires definition of the following small structure that we can guess (by iterating through every possibility) and later extend to find a solution.

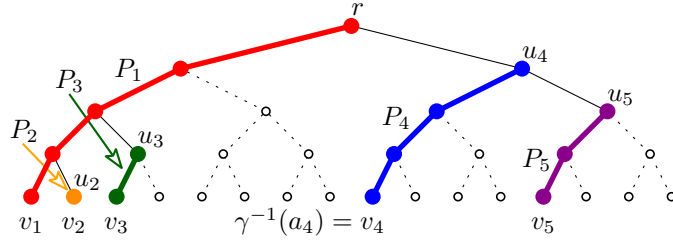
► **Definition 10.** For any N -seeding γ , the *blueprint* generated by γ with respect to D_1, \dots, D_m , denoted by \mathcal{T} , consists of

- a blueprint subtree $T' \subseteq T$. This is the subtree of T induced by the ancestors of leaves $\{\gamma^{-1}(a) \mid a \in V_A\}$, and
- m labelings of T' constructed by restricting ℓ_i to $V(T')$ for each $i \in [m]$.

We will abuse the notation in the context of blueprints, by using ℓ_i to denote the labelings restricted to T' . See Figure 3 for an example of a blueprint.

Note that for an instance of STF there are a total of $n!/2^{n-1}$ possible choices for the solution seeding. Our algorithm approaches this massive search space by instead finding a blueprint that preserves enough information from the full solution but is simple enough that there are not many of them. Several papers, such as [23, 9, 10, 11], use a similar approach where a substructure called a *template* is used. However, our approach differs significantly from these papers since our notion of a blueprint is based on the bracket and hence a complete binary tree representation of the outcome of the tournament. On the other hand, the previous papers working with FPT algorithms for TF have generally defined so-called templates using the notion of *spanning binomial arborescences* (SBA), which are a specific type of spanning trees. Indeed, TF has a well-established connection to SBAs [30], that states that there is a solution to the TF instance if and only if the tournament digraph has an SBA rooted at the favorite player. However, SBAs are unsuitable for our setting since they directly represent the winner of each match in their structure, in our case we would have to deal with multiple SBAs in parallel, which appears to be challenging, technically. As a result, the notion of blueprint considered here, Definition 10, is a significant deviation from the literature on FPT algorithms for this type of problems.

► **Observation 11.** *Every vertex that is not in the blueprint subtree has the same label in every ℓ_i . More precisely, for every $u \in V(T) \setminus V(T')$ we have $\ell_i(u) = \ell_j(u)$ for all $i, j \in [m]$.*



■ **Figure 3** Example of a blueprint subtree on a bracket with $n = 16$ and $|V_A| = 5$. The blueprint subtree consists of solid vertices and edges. Empty vertices and dotted edges are in the bracket, but not in the blueprint. The colored thick edges mark the blueprint partition into paths P_j for $j \in [5]$.

That is, the blueprint includes all of the information that changes between the different input tournament digraphs. We want to find blueprints without knowing the seeding that generates them. To do this we use the following lemma which specifies conditions under which $(T', \ell_1, \dots, \ell_m)$ constitutes a blueprint.

► **Lemma 12.** *Given m Types-digraphs F_i , a subtree $T' \subseteq T$, and m labelings $\ell_i: V(T') \rightarrow \text{Types}$, suppose:*

1. *for each $v \in V(T')$ with two children in T' , u and w , for each $i \in [m]$ we have $\ell_i(v) = \ell_i(u)$ if $\ell_i(u)\ell_i(w) \in E(F_i)$ and $\ell_i(v) = \ell_i(w)$ otherwise.*
2. *for each $v \in V(T')$ with exactly one child u in T' there exists a type $t \in \text{Flex}$ such that for every $i \in [m]$ let $q := \ell_i(u)$, either $qt \in E(F_i)$ and $\ell_i(v) = q$ or $tq \in E(F_i)$ and $\ell_i(v) = t$, and*
3. *every vertex v with no children in T' is also a leaf in T and $\ell_i(v) = \ell_j(v) \in V_A$ for all $i, j \in [m]$.*

Then there exist m tournament digraphs, D_1, \dots, D_m and an N -seeding γ such that $(T', \ell_1, \dots, \ell_m)$ is the blueprint generated by γ with respect to D_1, \dots, D_m and F_i is the Types-digraph generated by D_i for each $i \in [m]$.

Proof. We aim to construct a seeding γ and tournament digraphs D_1, \dots, D_m which generate our blueprint $(T', \ell_1, \dots, \ell_m)$. Due to Definition 10 and Condition 3 we know that $\gamma(v) = \ell_i(v)$ for all $v \in L(T')$ and $i \in [m]$, which fixes all affected vertices of the tournament digraphs. As the remaining leaves $Q = L(T) \setminus L(T')$ are not part of the blueprint, we know by Observation 11 that $\ell_i(v) = \ell_j(v)$ for all $i, j \in [m]$ and $v \in Q$. All singular types were assigned, hence, the leaves Q must be assigned players with flexible types. In particular, for every non-blueprint vertex w whose parent v is in the blueprint we can find a type $t \in \text{Flex}$ according to Condition 2. To ensure that vertex w indeed gets type t we take the set of all the leaves under w and for each of them $q \in Q \cap \text{Desc}(w)$ we create a new player p with type $\tau(p) = t$ and assign $\gamma(q) = t$. We see that this way each leaf $q \in Q$ gets a new player because when we follow a path from the leaf to the root then the first vertex of T' we encounter is v and the one before is w which when processed as described above assigned a player to q . Hence, γ is complete. Finally, for each $i \in [m]$ as $F_i = D_i^\tau$ we know that the singular types of F_i directly translate to players in D_i . The players that get flexible types were created when we devised γ . As the last step, we add arcs to D_i so that $uv \in E(D_i) \iff \tau(u)\tau(v) \in E(F_i)$ as implied by Observation 7. ◀

We want to find a bound on the number of blueprints so we can enumerate them in the desired time complexity. Towards that, we first show that the sequences of labels along paths from leaves to the root are well structured.

► **Lemma 13.** *Let $s_1, s_2, \dots, s_{\log n}$ be a sequence of types assigned by ℓ_i along a path in T starting in a leaf x up to the root r , i.e., $s_1 = \ell_i(x)$ and $s_{\log n} = \ell_i(r)$, then the number of positions $j \in [\log n - 1]$ such that $s_j \neq s_{j+1}$ is $2k(k+1)$.*

Proof. We take a tournament digraph D_i and observe, that every of its arcs is used at most once because such match implies one of its players was eliminated from the bracket. In our case, however, some types (τ) represent multiple vertices so arcs in D_i^r may be repeated if one of their endpoints is not a singleton type. We have \prec the ordering of the players that witnesses that the shared graph has FAS at most k . Consider one $i \in [\log n - 1]$ such that $s_i \neq s_{i+1}$. These types represent players x and y such that $\tau(x) = s_{i+1}$ and $\tau(y) = s_i$ and x won against y so we have arc $xy \in E(D)$. There are two possibilities, either the new type s_{i+1} is strictly right of s_i or strictly left of s_i with respect to the ordering \prec . In the strictly right case, say s_i is j -th type when we order the types from 1 up to $2k+1$ in the \prec ordering. As $s_i \prec s_{i+1}$ we know that s_{i+1} is at least $(j+1)$ -th in the \prec ordering. Looking at the whole sequence of type labels, the case where $s_i \prec s_{i+1}$ may repeat at most $2k$ times in a row. In the strictly left case, we have $s_{i+1} \prec s_i$ so xy is a back arc so this case may appear at most k times in total. For each left case we may have the right case repeating up to $2k$ times which gives us an upper bound on the number of positions where the type changes of $2k(k+1) \in \mathcal{O}(k^2)$. ◀

While there are $\mathcal{O}(n^n)$ N -seedings and even $\mathcal{O}(k^n)$ Types-seedings, there are only $f(k) \cdot n^{\mathcal{O}(1)}$ different blueprints.

► **Lemma 14.** *There exists a function f such that there are only $f(k) \cdot n^{\mathcal{O}(1)}$ blueprints. Additionally, it is possible to iterate through all of them that agree with a given sequence of Types-digraphs in FPT time.*

Proof. First, we partition the blueprint subtree T' into $|V_A|$ paths P_j in the following way, see Figure 3. Each path P_j has one endpoint in a leaf v_j that maps to an affected vertex $\gamma(v_j) = a_j$, P_1 has the other endpoint in root r , and all the other paths P_j for $j \geq 2$ end in a vertex u_j whose parent belongs to a different path $P_{j'}$, $j' \neq j$. These decompositions are characterized having for each $2 \leq j \leq |V_A|$ by height of u_j (i.e. length of P_j) and index of the path parent of u_j belongs to. Height of u_j is upper bounded by $\log n$ and we can upper bound possible indices of parent paths by $|V_A|$. This gives us an upper bound on the number of blueprint subtrees $(\log n \cdot |V_A|)^{|V_A|}$. Note we may be overcounting as one may decompose T' in multiple ways, however, decomposition gives a unique way to retrieve T' . So every possible T' is counted and the total number of blueprints is upper bounded by the number of decompositions.

Second, we need to upper bound the number of possible blueprint labelings ℓ_i for $i \in [m]$. Note that each P_j of the blueprint subtree decomposition is part of path P'_j that goes from v_j to the root r . Due to Lemma 13 we can upper bound the number of changes of ℓ_i along P'_j to $\mathcal{O}(k^2)$. Joining this bound over all $i \in [m]$ we get that there are no more than $\mathcal{O}(m \cdot k^2)$ changes (in any labeling) along P'_j . Hence, we can represent labelings of P'_j by $\mathcal{O}(m \cdot k^2)$ runs where each run is a tuple made of a run label and a run integer. Run labels have $(2k+1)^m$ possible values because they combine labels ℓ_i for all $i \in [m]$. Run integers say how many vertices have the specified labels in a row and are in $[\log n]$. Therefore, there are at most $((2k+1)^m \cdot \log n)^{\mathcal{O}(m \cdot k^2)}$ labelings of P'_j . Repeating this argument again for all paths P'_j , $j \in |V_A|$, we get that the blueprint subtree contains no more than $((2k+1)^m \cdot \log n)^{\mathcal{O}(m \cdot k^2 \cdot |V_A|)}$ vertices u that have a child v such that $\ell_i(u) \neq \ell_i(v)$ for some $i \in [m]$.

7:10 On Controlling Knockout Tournaments Without Perfect Information

Combining the above bounds we have at most $(\log n \cdot |V_A|)^{|V_A|} \cdot ((2k+1)^m \cdot \log n)^{\mathcal{O}(m \cdot k^2 \cdot |V_A|)}$ labeled blueprints. Finally, by the bounds $m \leq 2^k$ and $|V_A| \leq 2k+1$ we get $(k^{2^k} \cdot \log n)^{\mathcal{O}(2^k \cdot k^3)}$. Choosing $f(k) = k^{2^{k^2}}$ upper bounds the total number of possible blueprints by $f(k) \cdot n^{\mathcal{O}(1)}$ by Lemma 3.

To iterate through all the blueprints we first guess the parent of each path P_j and the length of each path before it reaches its parent path for $j \geq 2$. This gives a tree T' . We have $T' \subseteq T$ if and only if T' is binary so we check that every vertex has at most two children. Next we guess the labels for each P'_j by guessing $m \cdot 2k(k+1)$ runs. We then check that the guessed labels agree on every vertex that is shared between the P'_j s. Finally we check the conditions of Lemma 12. This tells us that our guess is a blueprint generated by γ . We have already shown that the number of guesses we could make is upper bounded by $f(k) \cdot n^{\mathcal{O}(1)}$. \blacktriangleleft

This shows we can guess the blueprint efficiently. Next, we take a blueprint \mathcal{T} and construct an ILP where the existence of a feasible solution is both a necessary and sufficient condition for the existence of a solution to the instance of STF with blueprint \mathcal{T} .

In the remainder of the paper we frequently use the following set of specific blueprint vertices. We also divide it into two sub-categories.

► **Definition 15 (Important vertices).** For a blueprint $\mathcal{T} = (T', \ell_1, \dots, \ell_m)$ the set of *important vertices* $\text{imp}(\mathcal{T})$ is the subset of vertices in $V(T) \setminus V(T')$ that have a parent from $V(T')$. I.e., these are the non-blueprint children of the blueprint vertices; note that a blueprint vertex cannot have two non-blueprint children.

For each important vertex $w \in \text{imp}(\mathcal{T})$, let v be the parent of w , and let u be the sibling of w (the other child of v). Note that T and T' agree on these relationships and $u, v \in V(T')$ while w is only in $V(T)$. If $\ell_i(v) = \ell_i(u)$ for all $i \in [m]$, we add (u, v, w) to $J_{\mathcal{T}}$. Otherwise there exists $i \in [m]$ such that $\ell_i(v) \neq \ell_i(u)$ and we add (u, v, w, i) to $K_{\mathcal{T}}$ for some such i .

The reason for these two categories is that wherever the type label changes at a vertex with only one child in the blueprint tree, the other child must be labeled with the new type (Lemma 12). This reduces the number of players needed to pack into this subtree by one.

ILP Feasibility

Given a blueprint $\mathcal{T} = (T', \ell_1, \dots, \ell_m)$, we initialize variables b_t and c_t for $t \in \text{Flex}$ where b_t keeps the number of leaves that need to be mapped to a type that either is equal to t or is beaten by t and c_t keeps the number of players of type t that remain to be assigned to the solution. We initially set $b_t = 0$ and $c_t = |\{j \in N : \tau(j) = t\}|$ for all $t \in \text{Flex}$.

We now consider each important vertex by iterating through $J_{\mathcal{T}}$ and $K_{\mathcal{T}}$ (in any order):

- For each $(u, v, w) \in J_{\mathcal{T}}$ we add $2^{\text{height}(w)}$ to b_q where q is the strongest type from Flex that gets beaten by all $\ell_1(v), \dots, \ell_m(v)$.
- For each $(u, v, w, i) \in K_{\mathcal{T}}$ we add $2^{\text{height}(w)} - 1$ to $b_{\ell_i(v)}$ and then decrement $c_{\ell_i(v)}$ by one. Recall that i in this case reflects $\ell_i(v) \neq \ell_i(u)$.

If $c_t < 0$ for any $t \in \text{Flex}$ then the blueprint requires more players of type t than are available, so we can safely reject it.

Now, we define our instance of ILP-Feasibility, denoted by $I_{\mathcal{T}}$, over a set of $\mathcal{O}(k^{m+1})$ non-negative constants b_s and c_t , and $\mathcal{O}(k^{m+1})$ variables $x_{s,t}$, where $s, t \in \text{Flex}$. The ILP has the following constraints.

$$\text{For all } t \in \text{Flex}, \quad \sum_{s \in \text{Flex}} x_{s,t} = c_t, \quad (1)$$

$$\text{For all } s \in \text{Flex}, \quad \sum_{t \in \text{Flex}} x_{s,t} = b_s, \quad (2)$$

$$\text{For all } s, t \in \text{Flex} \text{ with } t \prec s \quad x_{s,t} = 0, \quad (3)$$

$$\text{For all } s, t \in \text{Flex} \quad x_{s,t} \geq 0. \quad (4)$$

The intuition is as follows. Variable $x_{s,t}$ represents how many leaves we assigned to have type t that cannot be assigned a type stronger than s . Equation (1) ensures the number of players assigned type t is equal to the number of players we have available, Equation (2) ensures that each group of leaves with strongest type s is assigned the correct number of players, Equation (3) forbids assignment of a player that is too strong to a group, and we have Equation (4) so that we are assigning a non-negative number of players.

► **Lemma 16.** *Given an instance of STF that has a solution γ , there exists a blueprint, \mathcal{T} , such that $I_{\mathcal{T}}$ is feasible.*

Proof. We construct the blueprint generated by γ , and denote it $\mathcal{T} = (T', \ell_1, \dots, \ell_m)$. We will construct a solution to $I_{\mathcal{T}}$ as follows.

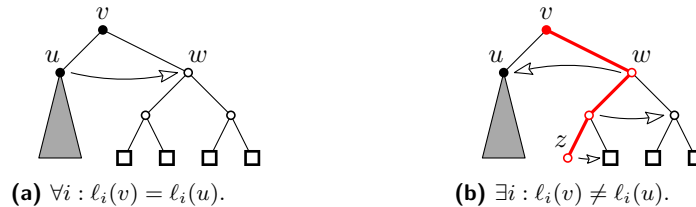
Initially set $x_{s,t} = 0$. Then we process the important vertices, see Definition 15, as follows.

- For each $(u, v, w) \in J_{\mathcal{T}}$ we have that $\ell_i(u)$ beats $\ell_i(w)$ for all $i \in [m]$. So for each $t \in \text{Flex}$ we add $|\{y \in \text{Desc}(w) \cap L(T) : \tau(\gamma(y)) = t\}|$ to $x_{q,t}$ where q is the strongest flexible type that gets beaten by all $\ell_1(v), \dots, \ell_m(v)$.
- For each $(u, v, w, i) \in K_{\mathcal{T}}$ we have that $\ell_i(w)$ beats $\ell_i(u)$. Even if this also occurs for some other i there is still only one new label by Observation 11 (the label of u could be different but there is only one label of w). We can trace back the type $\ell_i(w)$ to a leaf $z \in \text{Desc}(w) \cap L(T)$ where $\tau(\gamma(z)) = \ell_i(w)$. For each $t \in \text{Flex}$ we add $|\{y \in \text{Desc}(w) \cap L(T) \setminus \{z\} : \tau(\gamma(y)) = t\}|$ to $x_{\ell_i(w),t}$.

Effectively, we are taking the important vertices as shown in Definition 15 and assigning the values $x_{s,t}$ according to the seeding γ .

However, we also know that at least one leaf that is a descendant of a vertex where the label in the blueprint changes must be assigned a player with the type that the label changes to. We exclude these known leaves from the count. It remains to show that this assignment satisfies all the constraints.

We count each player exactly once except for $\gamma(z)$ for tuples in $K_{\mathcal{T}}$. Since c_t starts as the total number of players of type t and the appropriate c_t is decremented for each tuple in $K_{\mathcal{T}}$, the total number of players of type t we count is exactly c_t so Equation (1) is satisfied.



■ **Figure 4** Depiction of ℓ_i in cases during creation of the ILP instance. Arrows depict match where the player at arrow tail wins. Red bold path depicts the player that gets to v in ℓ_i . (a) u beats w so all leaves necessarily lose to u . (b) There is a leaf z that beats u and it also beats all other leaves.

7:12 On Controlling Knockout Tournaments Without Perfect Information

Every leaf of T that is not in the blueprint is a descendant of exactly one important vertex. Hence it counts towards exactly one b_s except for z for tuples in $K_{\mathcal{T}}$ which is excluded. Therefore the total number of leaves (regardless of the type of the player assigned to them by γ) that are descendants of w is $2^{\text{height}(w)}$ for tuples in $J_{\mathcal{T}}$ and one less for tuples in $K_{\mathcal{T}}$. Summing over all possible important vertices gives us b_s , so Equation (2) is satisfied.

We know that $\ell_i(x) \prec \ell_i(v)$ never happens for any $i \in [m]$ and $x \in \text{Desc}(w)$ by the definition of bracket and since every back arc is between affected vertices and x cannot be an affected vertex. In particular the flexible types of the leaves $y \in \text{Desc}(w) \cap L(T)$ cannot beat $\ell_i(v)$ for any $i \in [m]$. This means they will never contribute to any $x_{s,t}$ where $t \prec s$ and hence the assignment of $x_{s,t}$ will satisfy Equation (3). ◀

We now show the converse of Lemma 16.

► **Lemma 17.** *If there exists a blueprint \mathcal{T} where $I_{\mathcal{T}}$ has a solution, then the instance of STF is a yes-instance.*

Proof. We construct the solution γ by assigning players to $\gamma(v)$ for all $v \in L(T)$. For this proof, we define *assign player p to leaf v* as setting $\gamma(v) = x$, and marking x and v so that they may not be assigned later. The marking plays a role when we *assign set P to set L* which means we assign $|L|$ arbitrary players of P to arbitrary leaves in L one by one.

We start by assigning to the leaves of the blueprint. The labelings in any blueprint always agree on the leaves, i.e., for $v \in L(T')$ we have $\ell_i(v) = \ell_j(v)$ for all $i, j \in [m]$. So we can assign $\ell_i(v)$ to v for every $v \in L(T')$.

It remains to assign players to the other leaves of T . Note that all the remaining players have flexible types and they form a total order. Let B_s where $s \in \text{Flex}$ be called a *bag*, we will gradually add leaves to the bags. Each bag indicates what is the strongest flexible type s its leaves can be assigned to. Initially, we set all bags $B_s = \emptyset$. Our aim is to add the remaining leaves to the bags in a way that $|B_s| = b_s$. We process each important vertex as follows:

- For each $(u, v, w) \in J_{\mathcal{T}}$ we add $L(T) \cap \text{Desc}(w)$ to B_q where q is the strongest flexible type that gets beaten by all $\ell_1(v), \dots, \ell_m(v)$.
- For each $(u, v, w, i) \in K_{\mathcal{T}}$ we choose an arbitrary $z \in L(T) \cap \text{Desc}(w)$ and $j \in \tau^{-1}(\ell_i(v))$, assign j to z , then add $L(T) \cap \text{Desc}(w) \setminus \{z\}$ to $B_{\ell_i(v)}$.

Observe that for each tuple the number of vertices added to B_s is equal to the increase of b_s in definition of the ILP, hence, we have $|B_s| = b_s$ for each $s \in \text{Flex}$. Moreover, at this point the number of unassigned players of type $t \in \text{Flex}$ is equal to c_t because we started with all non-blueprint vertices and assigned exactly one for each tuple in $K_{\mathcal{T}}$ which reflects the decrease of c_t by one in the definition of the ILP.

For each $s, t \in \text{Flex}$ assign $x_{s,t}$ players of type t to B_s . We assign $x_{s,t}$ players of $\tau^{-1}(t)$ to the leaves B_s . Since $x_{s,t}$ is a solution to $I_{\mathcal{T}}$ we know by Equation (1) that we assign the right number of players to each bag, by Equation (2) we know that from each type we assign the remaining unassigned c_t players. Lastly by Equation (3) we know that the assigned players will lose as appropriate to players assigned to vertices of the blueprint (for tuples in $J_{\mathcal{T}}$) or z (for tuples in $K_{\mathcal{T}}$). Hence, a feasible ILP implies we have a yes-instance. ◀

Proof of Theorem 5. Given an instance of STF, $(D_1, \dots, D_m, \alpha^*)$, we first calculate the types and the Types-digraphs generated by each D_i . Then, according to Lemma 14, we can iterate through the feasible blueprints in FPT time. For each of these blueprints we prepare an ILP instance. We first initialize it in $\mathcal{O}(k^2)$ time. Then for each blueprint we calculate the $\mathcal{O}(n)$ elements of $J_{\mathcal{T}}$ and $K_{\mathcal{T}}$ and we perform $\mathcal{O}(1)$ operations on each element. The ILP contains Equations (1) and (2) $|\text{Flex}|$ times each. Each contains $|\text{Flex}|$ variables

and a constant. It also contains Equation (3) $|\text{Flex}| \cdot (|\text{Flex}| - 1)$ times and Equation (4) $|\text{Flex}|^2$ times each with one variable and one constant. All values are upper bounded by n so the total number of bits in the input is $\mathcal{O}(|\text{Flex}|^2 \cdot \log n)$ and we have $|\text{Flex}|^2$ variables. As $|\text{Flex}| = 2k + 1$ by Proposition 4 we solve the ILP instance in $\mathcal{O}((k^2)^{2.5k^2 + o(k^2)} \cdot k^2 \log n)$ time and space polynomial in $k^2 \cdot \log n$. If the ILP instance is feasible then, by Lemma 17, we answer yes. If we have iterated through every blueprint and none of the ILP instances were feasible we answer no by Lemma 16. ◀

4 Application to PTF

Let us define the parameterization we consider for PTF. Recall an instance of PTF is of the form (N, P, p^*, α^*) where P is a matrix of pairwise winning probabilities over the player set N and p^* is the target probability with which we want the player α^* to win.

► **Definition 18** (Parameter for PTF). Recall that for an instance (N, P, p^*, α^*) of PTF, the *certainty digraph* is the digraph defined over the vertex set N where there is an arc uv in the digraph if and only if $P_{u,v} = 1$. Define the *degree of uncertainty* of this instance as the number of pair sets $\{u, v\}$ from N such that $P_{u,v}$ is not equal to 0 or 1.

Notice that in the above definition, if (N, P, p^*, α^*) is an instance of PTF, then the degree of uncertainty is half the number of fractional values in P .

As a consequence of Theorem 5, we obtain the following algorithm for PTF.

► **Theorem 19.** PTF is FPT parameterized by the FAS number of the certainty digraph and the degree of uncertainty.

We observe that the shared digraph in STF is analogous to the certainty digraph in PTF. Hence, we prove Theorem 19 by proving the following lemma that reduces PTF to STF and then using the algorithm of Theorem 5 in the premise of Lemma 20.

► **Lemma 20.** If one can solve STF in time \mathcal{T} , then PTF can be solved in time $\mathcal{T} \cdot 2^{2^k} \cdot n^{\mathcal{O}(1)}$ where k is the degree of uncertainty of the PTF instance and n is the input size.

Proof. Let (N, P, α^*, p^*) denote an instance of PTF and let C be the certainty digraph.

Let us first sketch the idea behind the reduction. If we were to run the probabilistic experiment (thereby determining who wins each uncertain match) we would get a new arc between every pair of players u, v where neither uv nor vu are in C : the arc uv appears in the tournament with probability P_{uv} , otherwise the arc vu appears there instead. Our goal in the reduction is to consider all 2^k possible outcomes of this random process. They are tournament digraphs on N (i.e. “completions of the certainty digraph”), call them D_1, \dots, D_{2^k} , where $C \subseteq D_i$ for each $i \in [2^k]$. Note that, since the orientations of the arcs not in the certainty digraph are chosen independently, the probability that we would get D_i is just the product of each $P_{u,v}$ where $uv \in E(D_i) \setminus E(C)$. Since the tournament digraphs D_i are elementary events in our sample space, we have that the probability of an event $\mathcal{D} \subseteq \{D_1, \dots, D_{2^k}\}$ occurring is simply the sum of the probabilities of each $D_i \in \mathcal{D}$. So we can, completely deterministically, for each event \mathcal{D} , calculate its probability of occurrence.

Let us now return to the actual reduction. As described above, for each event \mathcal{D} , we calculate its probability of occurrence. If it is at least p^* we create a new STF instance comprising the player α^* and the digraphs in \mathcal{D} and solve the instance in \mathcal{T} time using the algorithm assumed in the premise.

If any of these instances is a yes-instance (i.e. there is a seeding γ where α^* wins the bracket generated by γ with respect to $D_i \in \mathcal{D}$ for every i), then we argue that this seeding is a solution for PTF as follows. The probability of at least one of the D_i s occurring is at least p^* and in each of them α^* wins the bracket generated by γ . Hence the probability of α^* winning the bracket generated by γ is at least p^* .

Conversely, if we have a solution seeding for the PTF instance then there is some non-empty collection of D_i s ($i \in [2^k]$) such that α^* wins in each. Call this collection \mathcal{D} . Since we started with a solution seeding, the probability of \mathcal{D} occurring (i.e., the sum of the probabilities of occurrence of the D_i s in \mathcal{D}) is at least p^* and hence this collection is one of the STF instances created in our reduction.

As there are 2^{2^k} possible events \mathcal{D} and for each, we perform a polynomial-time processing to construct the STF instance and then invoke the assumed algorithm that runs in \mathcal{T} time, we obtain $\mathcal{T} \cdot 2^{2^k} \cdot n^{\mathcal{O}(1)}$ time complexity for PTF. ◀

5 Concluding remarks and future work

We have obtained the first fixed-parameter tractability results for SE tournament design with potential imperfect information. The rich body of work on the deterministic version provides natural directions for future research: for example, a probabilistic version of parameterization with respect to feedback vertex set number, as studied by Zehavi [31] would be an improvement over this work. Alternately, probabilistic modeling of *demand tournament fixing* or *popular tournament fixing*, where the goal is to schedule “high-value” matches as opposed to ensuring a specific player wins, as studied by Gupta et al. [12] and Chaudhary et al. [5], respectively, are also possible research directions.

References

- 1 Haris Aziz, Serge Gaspers, Simon Mackenzie, Nicholas Mattei, Paul Stursberg, and Toby Walsh. Fixing balanced knockout and double elimination tournaments. *Artif. Intell.*, 262:1–14, 2018. doi:10.1016/J.ARTINT.2018.05.002.
- 2 John J. Bartholdi, Craig A. Tovey, and Michael A. Trick. The computational difficulty of manipulating an election. *Social Choice and Welfare*, 6(3):227–241, 1989. URL: <http://www.jstor.org/stable/41105918>.
- 3 John J. Bartholdi, Craig A. Tovey, and Michael A. Trick. How hard is it to control an election? *Mathematical and Computer Modelling*, 16(8):27–40, 1992. doi:10.1016/0895-7177(92)90085-Y.
- 4 Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel D. Procaccia. *Handbook of Computational Social Choice*. Cambridge University Press, USA, 1st edition, 2016.
- 5 Juhi Chaudhary, Hendrik Molter, and Meirav Zehavi. How to make knockout tournaments more popular? *CoRR*, abs/2309.09967, 2023. doi:10.48550/arXiv.2309.09967.
- 6 Juhi Chaudhary, Hendrik Molter, and Meirav Zehavi. Parameterized analysis of bribery in challenge the champ tournaments. *CoRR*, abs/2403.17587, 2024. doi:10.48550/arXiv.2403.17587.
- 7 Connolly and Rendleman. Tournament qualification, seeding and selection efficiency. *Technical Report 2011-96, Tuck School of Business*, 2011.
- 8 András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Comb.*, 7(1):49–65, 1987. doi:10.1007/BF02579200.
- 9 Sushmita Gupta, Sanjukta Roy, Saket Saurabh, and Meirav Zehavi. When rigging a tournament, let greediness blind you. In *IJCAI*, pages 275–281. ijcai.org, 2018. doi:10.24963/IJCAI.2018/38.
- 10 Sushmita Gupta, Sanjukta Roy, Saket Saurabh, and Meirav Zehavi. Winning a tournament by any means necessary. In *IJCAI*, pages 282–288. ijcai.org, 2018. doi:10.24963/IJCAI.2018/39.

- 11 Sushmita Gupta, Saket Saurabh, Ramanujan Sridharan, and Meirav Zehavi. On succinct encodings for the tournament fixing problem. In *IJCAI*, pages 322–328. ijcai.org, 2019. doi:10.24963/IJCAI.2019/46.
- 12 Sushmita Gupta, Ramanujan Sridharan, and Peter Strulo. An exercise in tournament design: When some matches must be scheduled. In *AAAI*, pages 9749–9756. AAAI Press, 2024. doi:10.1609/AAAI.V38I9.28833.
- 13 Noam Hazon, Paul E. Dunne, Sarit Kraus, and Michael J. Wooldridge. How to rig elections and competitions. In *Second International Workshop on Computational Social Choice (COMSOC)*, pages 301–312, 2008.
- 14 Jeff Horen and Raymond Riezman. Comparing draws for single elimination tournaments. *Oper. Res.*, 33(2):249–262, 1985. doi:10.1287/OPRE.33.2.249.
- 15 Hendrik W. Lenstra Jr. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983. doi:10.1287/MOOR.8.4.538.
- 16 Ravi Kannan. Minkowski’s convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, 1987. doi:10.1287/MOOR.12.3.415.
- 17 Michael P. Kim and Virginia Vassilevska Williams. Fixing tournaments for kings, chokers, and more. In *IJCAI*, pages 561–567. AAAI Press, 2015. URL: <http://ijcai.org/Abstract/15/085>.
- 18 Kathrin Konczak and Jérôme Lang. Voting procedures with incomplete preferences. In *IJCAI-05 Multidisciplinary Workshop on Advances in Preference Handling*, 2005.
- 19 Jérôme Lang, Maria Silvia Pini, Francesca Rossi, Kristen Brent Venable, and Toby Walsh. Winner determination in sequential majority voting. In *IJCAI*, pages 1372–1377, 2007. URL: <http://ijcai.org/Proceedings/07/Papers/221.pdf>.
- 20 J.-Francois Laslier. Tournament solutions and majority voting. *Springer-Verlag*, 1997.
- 21 Nicholas Mattei, Judy Goldsmith, Andrew Klapper, and Martin Mundhenk. On the complexity of bribery and manipulation in tournaments with uncertain information. *J. Appl. Log.*, 13(4):557–581, 2015. doi:10.1016/J.JAL.2015.03.004.
- 22 Maria Silvia Pini, Francesca Rossi, Kristen Brent Venable, and Toby Walsh. Possible and necessary winners in voting trees: majority graphs vs. profiles. In *AAMAS*, pages 311–318. IFAAMAS, 2011. URL: <http://portal.acm.org/citation.cfm?id=2030516&CFID=69153967&CFTOKEN=38069692>.
- 23 M. S. Ramanujan and Stefan Szeider. Rigging nearly acyclic tournaments is fixed-parameter tractable. In *AAAI*, pages 3929–3935. AAAI Press, 2017. doi:10.1609/AAAI.V31I1.11132.
- 24 Sherwin Rosen. Prizes and incentives in elimination tournaments. *The American Economic Review*, 76(4):701–715, 1986.
- 25 Tyrel Russell and Peter van Beek. An empirical study of seeding manipulations and their prevention. In *IJCAI*, pages 350–356. IJCAI/AAAI, 2011. doi:10.5591/978-1-57735-516-8/IJCAI11-068.
- 26 Isabelle Stanton and Virginia Vassilevska Williams. Manipulating stochastically generated single-elimination tournaments for nearly all players. In *WINE*, volume 7090 of *Lecture Notes in Computer Science*, pages 326–337. Springer, 2011. doi:10.1007/978-3-642-25510-6_28.
- 27 Isabelle Stanton and Virginia Vassilevska Williams. Rigging tournament brackets for weaker players. In *IJCAI*, pages 357–364. IJCAI/AAAI, 2011. doi:10.5591/978-1-57735-516-8/IJCAI11-069.
- 28 Tullock. Toward a theory of the rent-seeking society. *Texas A&M University Press*, 1980.
- 29 Thuc Vu, Alon Altman, and Yoav Shoham. On the complexity of schedule control problems for knockout tournaments. In *AAMAS (1)*, pages 225–232. IFAAMAS, 2009. URL: <https://dl.acm.org/citation.cfm?id=1558044>.
- 30 Virginia Vassilevska Williams. Fixing a tournament. In *AAAI*, pages 895–900. AAAI Press, 2010. doi:10.1609/AAAI.V24I1.7617.
- 31 Meirav Zehavi. Tournament fixing parameterized by feedback vertex set number is FPT. In *AAAI*, pages 5876–5883. AAAI Press, 2023. doi:10.1609/AAAI.V37I5.25728.

Kernelization for Orthogonality Dimension

Ishay Haviv 

School of Computer Science, The Academic College of Tel Aviv-Yaffo, Israel

Dror Rabinovich

School of Computer Science, The Academic College of Tel Aviv-Yaffo, Israel

Abstract

The orthogonality dimension of a graph over \mathbb{R} is the smallest integer d for which one can assign to every vertex a nonzero vector in \mathbb{R}^d such that every two adjacent vertices receive orthogonal vectors. For an integer d , the d -ORTHO-DIM $_{\mathbb{R}}$ problem asks to decide whether the orthogonality dimension of a given graph over \mathbb{R} is at most d . We prove that for every integer $d \geq 3$, the d -ORTHO-DIM $_{\mathbb{R}}$ problem parameterized by the vertex cover number k admits a kernel with $O(k^{d-1})$ vertices and bit-size $O(k^{d-1} \cdot \log k)$. We complement this result by a nearly matching lower bound, showing that for any $\varepsilon > 0$, the problem admits no kernel of bit-size $O(k^{d-1-\varepsilon})$ unless $\text{NP} \subseteq \text{coNP/poly}$. We further study the kernelizability of orthogonality dimension problems in additional settings, including over general fields and under various structural parameterizations.

2012 ACM Subject Classification Theory of computation \rightarrow Parameterized complexity and exact algorithms; Mathematics of computing \rightarrow Graph coloring; Mathematics of computing \rightarrow Combinatorial algorithms

Keywords and phrases Orthogonality dimension, Fixed-parameter tractability, Kernelization, Graph coloring

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.8

Related Version *Full Version:* <https://arxiv.org/abs/2408.08380>

Funding *Ishay Haviv:* Research supported by the Israel Science Foundation (grant No. 1218/20).

Dror Rabinovich: Research supported by the Israel Science Foundation (grant No. 1218/20).

Acknowledgements We are grateful to the anonymous reviewers for their constructive and helpful feedback.

1 Introduction

For a field \mathbb{F} and an integer d , a d -dimensional orthogonal representation of a graph $G = (V, E)$ over \mathbb{F} is an assignment of a vector $u_v \in \mathbb{F}^d$ with $\langle u_v, u_v \rangle \neq 0$ to each vertex $v \in V$, such that for every two adjacent vertices v and v' in G , it holds that $\langle u_v, u_{v'} \rangle = 0$. We consider here the standard inner product, defined for any two vectors $x, y \in \mathbb{F}^d$ by $\langle x, y \rangle = \sum_{i=1}^d x_i \cdot y_i$ with operations performed over the field \mathbb{F} . The orthogonality dimension of a graph G over \mathbb{F} , denoted by $\xi_{\mathbb{F}}(G)$, is the smallest integer d for which G admits a d -dimensional orthogonal representation over \mathbb{F} (see Definition 10 and Remark 11).

The notion of orthogonal representations over the real field \mathbb{R} was introduced in 1979 by Lovász [26], who used them to define the celebrated ϑ -function that was motivated by questions in information theory on the Shannon capacity of graphs. Over the years, orthogonal representations and the orthogonality dimension have found a variety of applications in several areas of research. In graph theory, orthogonal representations over the reals were used by Lovász, Saks, and Schrijver [28] to characterize connectivity properties of graphs (see also [27, Chapter 10]). In computational complexity, the orthogonality dimension over finite fields was related to lower bounds in circuit complexity by Codenotti, Pudlák, and Resta [13] (see also [19, 18]). Over the complex field, it was employed by de Wolf [15] to



© Ishay Haviv and Dror Rabinovich;

licensed under Creative Commons License CC-BY 4.0

19th International Symposium on Parameterized and Exact Computation (IPEC 2024).

Editors: Édouard Bonnet and Paweł Rzażewski; Article No. 8; pp. 8:1–8:17

Leibniz International Proceedings in Informatics



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

determine the quantum one-round communication complexity of promise equality problems (see also [9, 6, 7]). Additional notable applications from the area of information theory are related to index coding [3, 1], distributed storage [2], and hat-guessing games [30].

The question of the complexity of determining the orthogonality dimension of a given graph over a specified field was proposed in 1989 by Lovász et al. [28]. For a field \mathbb{F} and an integer d , consider the decision problem that given a graph G asks to decide whether $\bar{\xi}_{\mathbb{F}}(G) \leq d$. It is easy to see that the problem can be solved efficiently for $d \in \{1, 2\}$, because a graph G satisfies $\bar{\xi}_{\mathbb{F}}(G) \leq 1$ if and only if it is edgeless, and it satisfies $\bar{\xi}_{\mathbb{F}}(G) \leq 2$ if and only if it is bipartite. For every $d \geq 3$, however, it was shown by Peeters [29] in 1996 that the problem is NP-hard for every field \mathbb{F} . More recently, it was shown in [12] that for every sufficiently large integer d , it is NP-hard to distinguish graphs G that satisfy $\bar{\xi}_{\mathbb{F}}(G) \leq d$ from those satisfying $\bar{\xi}_{\mathbb{F}}(G) \geq 2^{(1-o(1)) \cdot d/2}$, provided that \mathbb{F} is either a finite field or \mathbb{R} .

Motivated by the diverse applications of orthogonality dimension, the present paper delves into the computational complexity of this graph quantity from the perspective of parameterized complexity. We study the decision problems associated with orthogonality dimension with respect to various structural parameterizations, with a particular attention dedicated to the vertex cover number parameterization. We exhibit fixed-parameter tractability results for such problems, along with upper and lower bounds on their kernelizability. Our approach draws its inspiration from prior work on the parameterized complexity of coloring problems, most notably by Jansen and Kratsch [23] (see also [22, Chapter 7]) and by Jansen and Pieterse [24]. In what follows, we provide an overview on relevant research on coloring problems and then turn to a description of our contribution. We use here standard notions from the area of parameterized complexity, whose definitions can be found in Section 2.4 (see also, e.g., [14, 17]).

Graph coloring is a cornerstone concept in graph theory that has been extensively studied from a computational point of view. For an integer q , a q -coloring of a graph G is an assignment of a color to each vertex of G from a set of q colors. The coloring is said to be proper if it assigns distinct colors to every two adjacent vertices in the graph. A graph G is called q -colorable if it admits a proper q -coloring, and the smallest integer q for which G is q -colorable is called the chromatic number of G and is denoted by $\chi(G)$. For an integer q , let q -COLORING denote the decision problem that given a graph G asks to decide whether $\chi(G) \leq q$. The COLORING problem is defined similarly, with the only, yet crucial, difference that the number of colors q is not fixed but forms part of the input. It is well known that the q -COLORING problem can be solved in polynomial time for $q \in \{1, 2\}$ and is NP-complete for every $q \geq 3$. This implies that the COLORING problem, parameterized by the number of colors q , is not fixed-parameter tractable unless $P = NP$.

The study of the parameterized complexity of coloring problems was initiated in 2003 by Cai [8], who proposed the following terminology. For a family of graphs \mathcal{G} and for an integer k , let $\mathcal{G} + kv$ denote the family of all graphs that can be obtained from a graph of \mathcal{G} by adding at most k vertices (with arbitrary neighborhoods). Equivalently, a graph $G = (V, E)$ lies in $\mathcal{G} + kv$ if there exists a set $X \subseteq V$ of size $|X| \leq k$, referred to as a modulator, such that the graph $G \setminus X$ obtained from G by removing the vertices of X lies in \mathcal{G} . For example, letting EMPTY denote the family of all edgeless graphs, the family $\text{EMPTY} + kv$ consists of all the graphs that admit a vertex cover of size at most k . For an integer q , the q -COLORING problem on $\mathcal{G} + kv$ graphs is the parameterized problem defined as follows.

Input: A graph $G = (V, E)$ and a set $X \subseteq V$ such that $G \setminus X \in \mathcal{G}$.

Question: Is $\chi(G) \leq q$?

Parameter: The size $k = |X|$ of the modulator X .

As before, the COLORING problem on $\mathcal{G} + kv$ graphs is defined similarly, except that the number of colors q is not fixed but forms part of the input.

A common parameterization of graph problems that received a considerable amount of attention in the literature is that of the vertex cover number, corresponding to the family $\mathcal{G} = \text{EMPTY}$ (see, e.g., [16]). It is well known that the COLORING problem on $\text{EMPTY} + kv$ graphs is fixed-parameter tractable. Nevertheless, Bodlaender, Jansen, and Kratsch [5] proved that the problem does not admit a kernel of polynomial size under the assumption $\text{NP} \not\subseteq \text{coNP/poly}$, whose refutation is known to imply the collapse of the polynomial-time hierarchy [32]. Yet, for any fixed integer $q \geq 3$, Jansen and Kratsch [23] showed that the q -COLORING problem on $\text{EMPTY} + kv$ graphs admits a kernel with $O(k^q)$ vertices which can be encoded in $O(k^q)$ bits (see also [16]). This result was improved by Jansen and Pieterse [24] as an application of an algebraic sparsification technique they introduced in [25]. It was shown in [24] that for every $q \geq 3$, the q -COLORING problem on $\text{EMPTY} + kv$ graphs admits a kernel with $O(k^{q-1})$ vertices and bit-size $O(k^{q-1} \cdot \log k)$ (see [24] for various generalizations). On the contrary, it was shown in [23, 24] that for every $q \geq 3$ and any $\varepsilon > 0$, the problem does not admit a kernel that can be encoded in $O(k^{q-1-\varepsilon})$ bits unless $\text{NP} \subseteq \text{coNP/poly}$, thereby settling its kernelization complexity up to a multiplicative $k^{o(1)}$ term.

The paper [23] further studied the kernelization complexity of the q -COLORING problem on $\mathcal{G} + kv$ graphs for general families \mathcal{G} . In particular, they considered graph families \mathcal{G} that are hereditary (i.e., closed under removal of vertices) and that are, roughly speaking, local with respect to the q -LIST COLORING problem, in the sense that a NO instance of q -LIST COLORING that involves a graph from \mathcal{G} must have a NO sub-instance whose size depends solely on q . For such families \mathcal{G} , it was shown in [23] that the q -COLORING problem on $\mathcal{G} + kv$ graphs admits a kernel of polynomial size, and this result was complemented with a lower bound on the kernel size relying on the assumption $\text{NP} \not\subseteq \text{coNP/poly}$. These results apply, for example, for the families USPLIT and UCOCHORDAL of the graphs whose connected components are split graphs and chordal graphs respectively. On the other hand, strengthening a result of Bodlaender et al. [4], the authors of [23] proved that the 3-COLORING problem on $\text{PATH} + kv$ graphs does not admit a kernel of polynomial size unless $\text{NP} \subseteq \text{coNP/poly}$, where PATH stands for the family of path graphs.

1.1 Our Contribution

This paper initiates a systematic study of the parameterized complexity of the orthogonality dimension of graphs. It is noteworthy that the orthogonality dimension of graphs is related to their chromatic number. Indeed, for every field \mathbb{F} and for every graph G , it holds that $\bar{\xi}_{\mathbb{F}}(G) \leq \chi(G)$, because a proper q -coloring of G may be viewed as a q -dimensional orthogonal representation of G over \mathbb{F} that uses only vectors from the standard basis of \mathbb{F}^q (see Claim 12). Yet, it turns out that the two graph quantities can differ substantially, as there exist graphs where the orthogonality dimension is exponentially smaller than the chromatic number (see, e.g., [20, Proposition 2.2]). Our investigation of the parameterized complexity of orthogonality dimension problems aligns with the approach of [23, 24] for studying coloring problems within the parameterized complexity framework. While coloring problems are primarily combinatorial in nature, the attempt to prove analogous results for orthogonality dimension raises intriguing questions reflecting the algebraic aspects of this graph quantity.

We first introduce the decision problems associated with orthogonality dimension.

► **Definition 1.** For a field \mathbb{F} , the $\text{ORTHO-DIM}_{\mathbb{F}}$ problem is defined as follows.

Input: A graph $G = (V, E)$ and an integer d .

Question: Is $\bar{\xi}_{\mathbb{F}}(G) \leq d$?

For a field \mathbb{F} and a family of graphs \mathcal{G} , the (parameterized) $\text{ORTHO-DIM}_{\mathbb{F}}$ problem on $\mathcal{G} + kv$ graphs is defined as follows.

8:4 Kernelization for Orthogonality Dimension

Input: A graph $G = (V, E)$, a set $X \subseteq V$ such that $G \setminus X \in \mathcal{G}$, and an integer d .

Question: Is $\bar{\xi}_{\mathbb{F}}(G) \leq d$?

Parameter: The size $k = |X|$ of the modulator X .

For a field \mathbb{F} , an integer d , and a family of graphs \mathcal{G} , the (parameterized) d -ORTHO-DIM $_{\mathbb{F}}$ problem on $\mathcal{G} + kv$ graphs is defined as follows.

Input: A graph $G = (V, E)$ and a set $X \subseteq V$ such that $G \setminus X \in \mathcal{G}$.

Question: Is $\bar{\xi}_{\mathbb{F}}(G) \leq d$?

Parameter: The size $k = |X|$ of the modulator X .

Let us stress that the integer d forms part of the input in the ORTHO-DIM $_{\mathbb{F}}$ problem, whereas it is a fixed constant in the d -ORTHO-DIM $_{\mathbb{F}}$ problem. Note that the hardness result of [29] implies that for every field \mathbb{F} , the ORTHO-DIM $_{\mathbb{F}}$ problem parameterized by the solution value d is not fixed-parameter tractable unless $\text{P} = \text{NP}$.

The main parameterization we consider is the vertex cover number of the input graph, which corresponds to the family $\mathcal{G} = \text{EMPTY}$. We start with the following fixed-parameter tractability result.

► **Theorem 2.** *Let \mathbb{F} be either a finite field or \mathbb{R} . The ORTHO-DIM $_{\mathbb{F}}$ problem on $\text{EMPTY} + kv$ graphs is fixed-parameter tractable.*

In fact, we prove an extension of Theorem 2, showing that if the ORTHO-DIM $_{\mathbb{F}}$ problem over a field \mathbb{F} is decidable, then the corresponding ORTHO-DIM $_{\mathbb{F}}$ problem on $\text{EMPTY} + kv$ graphs is fixed-parameter tractable (see Theorem 14). While it is easy to see that the ORTHO-DIM $_{\mathbb{F}}$ problem is decidable for any finite field \mathbb{F} , the real case relies on a result of Tarski [31] on the decidability of the existential theory of the reals (see Proposition 17).

We next consider the kernelizability of the d -ORTHO-DIM $_{\mathbb{F}}$ problem parameterized by the vertex cover number for a fixed integer d . For finite fields \mathbb{F} , one may deduce from a result of [24] that the problem admits a kernel of polynomial size, where the degree of the polynomial grows exponentially with d . We prove the following generalized and stronger result.

► **Theorem 3.** *For every field \mathbb{F} and for every integer $d \geq 3$, the d -ORTHO-DIM $_{\mathbb{F}}$ problem on $\text{EMPTY} + kv$ graphs admits a kernel with $O(k^d)$ vertices and bit-size $O(k^d)$.*

Theorem 3 prompts us to determine the smallest possible kernel size for the d -ORTHO-DIM $_{\mathbb{F}}$ problem on $\text{EMPTY} + kv$ graphs. The following result furnishes a lower bound, conditioned on the complexity-theoretic assumption $\text{NP} \not\subseteq \text{coNP/poly}$.

► **Theorem 4.** *For every field \mathbb{F} , every integer $d \geq 3$, and any real $\varepsilon > 0$, the d -ORTHO-DIM $_{\mathbb{F}}$ problem on $\text{EMPTY} + kv$ graphs does not admit a kernel with bit-size $O(k^{d-1-\varepsilon})$ unless $\text{NP} \subseteq \text{coNP/poly}$.*

The proof of Theorem 4 combines the lower bound on kernels for coloring problems proved in [23, 24] with a novel linear-parameter transformation from those problems to those associated with orthogonality dimension. More specifically, we show that for every field \mathbb{F} and for every integer $d \geq 3$, it is possible to efficiently transform a graph G into a graph G' so that $\chi(G) \leq d$ if and only if $\bar{\xi}_{\mathbb{F}}(G') \leq d$ while essentially preserving the vertex cover number (see Theorem 25). This is in contrast to a reduction of [29], which is appropriate only for $d = 3$ and significantly increases the vertex cover number. The transformation relies on a gadget graph that enforces the vectors assigned to two specified vertices to be either orthogonal or equal up to scalar multiplication (see Lemma 21).

We remark that Theorem 4 implies that, unless $\text{NP} \subseteq \text{coNP}/\text{poly}$, the degree of the polynomial lower bound on the size of a kernel for the d -ORTHO-DIM $_{\mathbb{F}}$ problem on $\text{EMPTY} + kv$ graphs can be arbitrarily large when d grows. This yields that the ORTHO-DIM $_{\mathbb{F}}$ problem on $\text{EMPTY} + kv$ graphs, in which d constitutes part of the input, is unlikely to admit a kernel of polynomial size.

Theorems 3 and 4 leave a multiplicative gap of roughly k between the upper and lower bounds on the kernel size achievable for the d -ORTHO-DIM $_{\mathbb{F}}$ problem parameterized by the vertex cover number. For the real field \mathbb{R} , we narrow this gap to a multiplicative term of $k^{o(1)}$, as stated below.

► **Theorem 5.** *For every integer $d \geq 3$, the d -ORTHO-DIM $_{\mathbb{R}}$ problem on $\text{EMPTY} + kv$ graphs admits a kernel with $O(k^{d-1})$ vertices and bit-size $O(k^{d-1} \cdot \log k)$.*

The proof of Theorem 5 borrows the sparsification technique of [25] (see also [24]). A key technical ingredient in applying this method lies in a construction of a low-degree polynomial, which assesses the feasibility of assigning a vector to a vertex based on the vectors of its neighbors (see Lemma 20). Our construction hinges on the fact that the zero vector is the only self-orthogonal vector over the reals. It would be interesting to decide whether or not a similar upper bound on the kernel size could be obtained for finite fields, where this property does not hold.

We finally turn to the study of kernels for the d -ORTHO-DIM $_{\mathbb{F}}$ problem on $\mathcal{G} + kv$ graphs for general hereditary graph families \mathcal{G} . Our first result in this context offers a sufficient condition on \mathcal{G} for the existence of a polynomial size kernel for d -ORTHO-DIM $_{\mathbb{F}}$ on $\mathcal{G} + kv$ graphs. This condition is related to a variant of the d -ORTHO-DIM $_{\mathbb{F}}$ problem, termed d -SUBSPACE CHOOSABILITY $_{\mathbb{F}}$, which was previously studied in various forms (see, e.g., [21, 11]) and may be viewed as a counterpart of the q -LIST COLORING problem for orthogonal representations. In this problem, the input consists of a graph G and an assignment of a subspace of \mathbb{F}^d to each vertex, and the goal is to decide whether G admits an orthogonal representation over \mathbb{F} that assigns to every vertex a vector from its subspace. We show that if the d -SUBSPACE CHOOSABILITY $_{\mathbb{F}}$ problem on graphs from a family \mathcal{G} is local, in the sense that every NO instance has a NO sub-instance on at most $g(d)$ vertices, then the d -ORTHO-DIM $_{\mathbb{F}}$ problem on $\mathcal{G} + kv$ graphs admits a kernel with $O(k^{d \cdot g(d)})$ vertices. We demonstrate the applicability of this result for the graph families USPLIT and UCOCHORDAL . On the contrary, for the PATH family, we show that it is unlikely that the d -ORTHO-DIM $_{\mathbb{F}}$ problem on $\text{PATH} + kv$ graphs admits a polynomial size kernel even for $d = 3$.

1.2 Outline

The remainder of the paper is structured as follows. In Section 2, we collect several definitions and facts that will be used throughout the paper. In Section 3, we study the fixed-parameter tractability of the ORTHO-DIM $_{\mathbb{F}}$ problem parameterized by the vertex cover number and prove Theorem 2. In Section 4, we present polynomial size kernels for the d -ORTHO-DIM $_{\mathbb{F}}$ problem parameterized by the vertex cover number and prove Theorems 3 and 5. In Section 5, we complement the results of Section 4 by providing limits on the kernelizability of the d -ORTHO-DIM $_{\mathbb{F}}$ problem parameterized by the vertex cover number and prove Theorem 4. For our study on the kernelizability of the d -ORTHO-DIM $_{\mathbb{F}}$ problem on $\mathcal{G} + kv$ graphs for general hereditary graph families \mathcal{G} , we refer the reader to the full version of the paper.

2 Preliminaries

2.1 Notations

For an integer n , let $[n] = \{1, 2, \dots, n\}$. All graphs considered in this paper are simple. For a graph $G = (V, E)$ and a set $X \subseteq V$, we let $G[X]$ denote the subgraph of G induced by X . The set X is called a vertex cover of G if every edge of G is incident with a vertex of X . We let $G \setminus X$ denote the graph obtained from G by removing the vertices of X (and the edges that touch them). For a vertex $v \in V$, we let $N_G(v)$ denote the set of neighbors of v in G .

2.2 Linear Algebra

For a field \mathbb{F} and an integer d , two vectors $x, y \in \mathbb{F}^d$ are said to be orthogonal if $\langle x, y \rangle = 0$, where $\langle x, y \rangle = \sum_{i=1}^d x_i y_i$ with operations over \mathbb{F} . If $\langle x, x \rangle = 0$ then x is self-orthogonal, and otherwise it is non-self-orthogonal. The orthogonal complement of a subspace $W \subseteq \mathbb{F}^d$ is the subspace W^\perp of all vectors in \mathbb{F}^d that are orthogonal to all vectors of W , that is, $W^\perp = \{x \in \mathbb{F}^d \mid \forall y \in W, \langle x, y \rangle = 0\}$. Note that the orthogonal complement satisfies $\dim(W) + \dim(W^\perp) = d$ and $W = (W^\perp)^\perp$. The following simple lemma, proved in the full version of the paper, characterizes the subspaces whose orthogonal complement includes a non-self-orthogonal vector. Recall that the characteristic of a field is the smallest positive number of copies of the field's identity element that sum to zero, or 0 if no such number exists.

► **Lemma 6.** *Let \mathbb{F} be a field, let d be an integer, and let W be a subspace of \mathbb{F}^d .*

1. *If the characteristic of \mathbb{F} is 2, then there exists a non-self-orthogonal vector in W^\perp if and only if the all-one vector does not lie in W .*
2. *If the characteristic of \mathbb{F} is not 2, then there exists a non-self-orthogonal vector in W^\perp if and only if $W^\perp \not\subseteq W$.*

Borrowing the terminology of [25], we say that a field \mathbb{F} is efficient if field operations and Gaussian elimination can be performed in polynomial time in the size of a reasonable input encoding. All finite fields, as well as the real field \mathbb{R} when restricted to rational numbers (to ensure finite representation), are efficient.

► **Lemma 7.** *For every efficient field \mathbb{F} , there exists a polynomial-time algorithm that given a collection of vectors in \mathbb{F}^d , decides whether there exists a non-self-orthogonal vector in \mathbb{F}^d that is orthogonal to all of them.*

Proof. For input vectors $u_1, \dots, u_\ell \in \mathbb{F}^d$, let $W = \text{span}(u_1, \dots, u_\ell)$. Observe that there exists a non-self-orthogonal vector in \mathbb{F}^d that is orthogonal to u_1, \dots, u_ℓ if and only if there exists a non-self-orthogonal vector in the orthogonal complement W^\perp . By Lemma 6, for a field \mathbb{F} of characteristic 2, this is equivalent to the all-one vector not lying in W , and for every other field \mathbb{F} , this is equivalent to $W^\perp \not\subseteq W$ (that is, at least one vector of a basis of W^\perp does not lie in W). Since \mathbb{F} is an efficient field, these conditions can be checked in polynomial time. This completes the proof. ◀

For a field \mathbb{F} and an integer d , if W is a subspace of \mathbb{F}^d of dimension smaller than d , then its orthogonal complement W^\perp has dimension at least 1, hence there exists a nonzero vector orthogonal to W . However, if we require this vector not only to be nonzero but also non-self-orthogonal, its existence is no longer guaranteed. This consideration motivates the following definition.

► **Definition 8.** For a field \mathbb{F} and an integer d , let $m(\mathbb{F}, d)$ denote the largest integer m such that for every subspace W of \mathbb{F}^d with $\dim(W) < m$, there exists a non-self-orthogonal vector in W^\perp .

► **Remark 9.** For every field \mathbb{F} and for every integer $d \geq 1$, it holds that $1 \leq m(\mathbb{F}, d) \leq d$. Indeed, the lower bound holds because there exists a non-self-orthogonal vector orthogonal to the zero subspace of \mathbb{F}^d , and the upper bound holds because no nonzero vector is orthogonal to the entire vector space \mathbb{F}^d . For a field \mathbb{F} of characteristic 2, it holds that $m(\mathbb{F}, d) = 1$, because no non-self-orthogonal vector is orthogonal to the 1-dimensional subspace spanned by the all-one vector. For every other field \mathbb{F} , it holds that $m(\mathbb{F}, d) \geq \lceil d/2 \rceil$. To see this, consider a subspace $W \subseteq \mathbb{F}^d$ of dimension $\dim(W) < \lceil d/2 \rceil$, and observe that it satisfies $\dim(W^\perp) = d - \dim(W) > d - \lceil d/2 \rceil = \lfloor d/2 \rfloor$, and thus $\dim(W^\perp) > \dim(W)$. This implies that $W^\perp \not\subseteq W$, hence by Item 2 of Lemma 6, there exists a non-self-orthogonal vector in W^\perp , as required. We also observe that if the vector space \mathbb{F}^d has no nonzero self-orthogonal vectors, then $m(\mathbb{F}, d) = d$, because for every subspace $W \subseteq \mathbb{F}^d$ of dimension smaller than d there exists a nonzero vector in W^\perp . In particular, for every integer d , it holds that $m(\mathbb{R}, d) = d$.

2.3 Orthogonality Dimension

The orthogonality dimension of a graph over a given field is defined as follows.

► **Definition 10.** For a field \mathbb{F} and an integer d , a d -dimensional orthogonal representation of a graph $G = (V, E)$ over \mathbb{F} is an assignment of a vector $u_v \in \mathbb{F}^d$ with $\langle u_v, u_v \rangle \neq 0$ to each vertex $v \in V$, such that for every two adjacent vertices v and v' in G , it holds that $\langle u_v, u_{v'} \rangle = 0$. The orthogonality dimension of a graph G over a field \mathbb{F} , denoted by $\bar{\xi}_{\mathbb{F}}(G)$, is the smallest integer d for which G admits a d -dimensional orthogonal representation over \mathbb{F} .

► **Remark 11.** Let us emphasize that the definition of an orthogonal representation does not require vectors assigned to non-adjacent vertices to be non-orthogonal. Orthogonal representations that satisfy this additional property are called faithful (see, e.g., [27, Chapter 10]). Note that orthogonal representations of graphs are sometimes defined in the literature as orthogonal representations of the complement, requiring vectors associated with non-adjacent vertices to be orthogonal (with no constraint imposed on vectors of adjacent vertices). We decided to use here the other definition, but one may view the notation $\bar{\xi}_{\mathbb{F}}(G)$ as standing for $\xi_{\mathbb{F}}(\bar{G})$.

▷ **Claim 12.** For every field \mathbb{F} and for every graph G , it holds that $\bar{\xi}_{\mathbb{F}}(G) \leq \chi(G)$.

Proof. For a graph $G = (V, E)$, let $q = \chi(G)$, and consider a proper coloring $c : V \rightarrow [q]$ of G . Assign to each vertex $v \in V$ the vector $e_{c(v)}$ in \mathbb{F}^q , where e_i stands for the vector of \mathbb{F}^q with 1 on the i th entry and 0 everywhere else. The vectors assigned here to the vertices of G are obviously non-self-orthogonal vectors of \mathbb{F}^q . Further, for every two adjacent vertices v and v' in G , it holds that $c(v) \neq c(v')$, hence $\langle e_{c(v)}, e_{c(v')} \rangle = 0$. This implies that there exists a q -dimensional orthogonal representation of G over \mathbb{F} , and thus $\bar{\xi}_{\mathbb{F}}(G) \leq q$. ◁

2.4 Parameterized Complexity

We present here a few fundamental definitions from the area of parameterized complexity. For a thorough introduction to the field, the reader is referred to, e.g., [14, 17].

A parameterized problem is a set $Q \subseteq \Sigma^* \times \mathbb{N}$ for some finite alphabet Σ . A fixed-parameter algorithm for Q is an algorithm that given an instance $(x, k) \in \Sigma^* \times \mathbb{N}$ decides whether $(x, k) \in Q$ in time $f(k) \cdot |x|^c$ for some computable function f and some constant c . If Q admits a fixed-parameter algorithm, then we say that Q is fixed-parameter tractable.

A compression (also known as generalized kernel and bikernel) for a parameterized problem $Q \subseteq \Sigma^* \times \mathbb{N}$ into a parameterized problem $Q' \subseteq \Sigma^* \times \mathbb{N}$ is an algorithm that given an instance $(x, k) \in \Sigma^* \times \mathbb{N}$ returns in time polynomial in $|x| + k$ an instance $(x', k') \in \Sigma^* \times \mathbb{N}$, such that $(x, k) \in Q$ if and only if $(x', k') \in Q'$, and in addition, $|x'| + k' \leq h(k)$ for some computable function h . The function h is referred to as the size of the compression. If h is polynomial, then the compression is called a polynomial compression. If $|\Sigma| = 2$, the function h is called the bit-size of the compression. When we say that a parameterized problem Q admits a compression of size h , we mean that there exists a compression of size h for Q into some parameterized problem. A compression for a parameterized problem Q into itself is called a kernelization for Q (or simply a kernel). It is well known that a decidable problem admits a kernel if and only if it is fixed-parameter tractable.

A transformation from a parameterized problem $Q \subseteq \Sigma^* \times \mathbb{N}$ into a parameterized problem $Q' \subseteq \Sigma^* \times \mathbb{N}$ is an algorithm that given an instance $(x, k) \in \Sigma^* \times \mathbb{N}$ returns in time polynomial in $|x| + k$ an instance $(x', k') \in \Sigma^* \times \mathbb{N}$, such that $(x, k) \in Q$ if and only if $(x', k') \in Q'$, and in addition, $k' \leq h(k)$ for some computable function h . If h is polynomial, the transformation is called polynomial-parameter, and if h is linear, the transformation is called linear-parameter.

3 Fixed-Parameter Tractability of Ortho-Dim $_{\mathbb{F}}$

In this section, we prove that the ORTHO-DIM $_{\mathbb{F}}$ problem parameterized by the vertex cover number is fixed-parameter tractable for various fields \mathbb{F} (recall Definition 1).

3.1 Finite Fields

We begin with the simple case, where the field \mathbb{F} is finite. The proof resembles the one of the fixed-parameter tractability of the COLORING problem parameterized by the vertex cover number.

► **Theorem 13.** *For every finite field \mathbb{F} , ORTHO-DIM $_{\mathbb{F}}$ on EMPTY + kv graphs is fixed-parameter tractable.*

Proof. Fix a finite field \mathbb{F} . The input of ORTHO-DIM $_{\mathbb{F}}$ on EMPTY + kv graphs consists of a graph $G = (V, E)$, a vertex cover $X \subseteq V$ of G of size $|X| = k$, and an integer d . Consider the algorithm that given such an input acts as follows. If $d > k$ then the algorithm accepts. Otherwise, the algorithm enumerates all possible assignments of non-self-orthogonal vectors from \mathbb{F}^d to the vertices of X . For every such assignment, the algorithm checks for every vertex $v \in V \setminus X$ if there exists a non-self-orthogonal vector in \mathbb{F}^d that is orthogonal to the vectors assigned to the neighbors of v (note that they all lie in X). If there exists an assignment to the vertices of X such that the answer is positive for all the vertices of $V \setminus X$, then the algorithm accepts, and otherwise it rejects.

For correctness, observe first that the input graph G is $(k + 1)$ -colorable, as follows by assigning k distinct colors to the vertices of the vertex cover X and another color to the vertices of the independent set $V \setminus X$. It thus follows, using Claim 12, that $\bar{\xi}_{\mathbb{F}}(G) \leq \chi(G) \leq k + 1$. Therefore, if $d > k$, then it holds that $\bar{\xi}_{\mathbb{F}}(G) \leq d$, hence our algorithm correctly accepts. Otherwise, the algorithm tries all possible assignments of non-self-orthogonal vectors of \mathbb{F}^d to the vertices of X . Since the vertices of $V \setminus X$ form an independent set in G , an assignment to

the vertices of X can be extended to the whole graph if and only if for each vertex $v \in V \setminus X$ there exists a non-self-orthogonal vector in \mathbb{F}^d that is orthogonal to the vectors assigned to the neighbors of v (which all lie in X). Since this condition is checked by the algorithm for all possible assignments to the vertices of X , its answer is correct.

We finally analyze the running time of the algorithm. On instances with $d > k$, the algorithm is clearly efficient. For instances with $d \leq k$, the number of assignments of vectors from \mathbb{F}^d to the vertices of X is at most $|\mathbb{F}|^{d \cdot |X|} \leq |\mathbb{F}|^{k^2}$. Further, by Lemma 7, given a collection of vectors of \mathbb{F}^d , it is possible to decide in polynomial time whether there exists a non-self-orthogonal vector in \mathbb{F}^d that is orthogonal to all of them. This implies that our algorithm for $\text{ORTHO-DIM}_{\mathbb{F}}$ on $\text{EMPTY} + kv$ graphs can be implemented in time $|\mathbb{F}|^{k^2} \cdot n^{O(1)}$, where n stands for the input size, hence the problem is fixed-parameter tractable. ◀

3.2 General Fields

We turn to the following generalization of Theorem 13.

► **Theorem 14.** *Let \mathbb{F} be a field for which the $\text{ORTHO-DIM}_{\mathbb{F}}$ problem is decidable. Then the $\text{ORTHO-DIM}_{\mathbb{F}}$ problem on $\text{EMPTY} + kv$ graphs is fixed-parameter tractable.*

Recall that the algorithm of Theorem 13 for the $\text{ORTHO-DIM}_{\mathbb{F}}$ problem on $\text{EMPTY} + kv$ graphs enumerates all possible assignments of non-self-orthogonal vectors to the vertices of a given vertex cover. This approach is clearly not applicable when the field \mathbb{F} is infinite. In order to extend the fixed-parameter tractability result to general fields and to obtain Theorem 14, we use the following definition inspired by an idea of [23].

► **Definition 15.** *Let $G = (V, E)$ be a graph, let $X \subseteq V$ be a vertex cover of G , and let $d \geq m \geq 1$ be integers. We define the graph $\mathbf{K} = \mathbf{K}(G, X, m, d)$ as follows. We start with $\mathbf{K} = G[X]$. Then, for every subset $S \subseteq X$ of size $m \leq |S| \leq d$, if there exists a vertex $v \in V \setminus X$ such that $S \subseteq N_G(v)$, then we add to \mathbf{K} a new vertex v_S and connect it to all the vertices of S .*

The following lemma lists useful properties of the graph given in Definition 15 (recall Definition 8).

► **Lemma 16.** *Let $G = (V, E)$ be a graph, let $X \subseteq V$ be a vertex cover of G of size $|X| = k$, let $d \geq m \geq 1$ be integers, and let $\mathbf{K} = \mathbf{K}(G, X, m, d)$.*

1. *The set X forms a vertex cover of \mathbf{K} .*
2. *The number of vertices in \mathbf{K} is at most $k + \sum_{i=m}^d \binom{k}{i}$.*
3. *The graph \mathbf{K} can be encoded in $\binom{k}{2} + \sum_{i=m}^d \binom{k}{i}$ bits.*
4. *For every field \mathbb{F} with $m \leq m(\mathbb{F}, d)$, it holds that $\bar{\xi}_{\mathbb{F}}(\mathbf{K}) \leq d$ if and only if $\bar{\xi}_{\mathbb{F}}(G) \leq d$.*

Proof. Consider the graph $\mathbf{K} = \mathbf{K}(G, X, m, d)$ given in Definition 15. Since X is a vertex cover of G , it immediately follows from the definition that every edge of \mathbf{K} is incident with a vertex of X , hence X is a vertex cover of \mathbf{K} , as required for Item 1. It further follows that the vertex set of \mathbf{K} consists of the vertices of X and at most one vertex per every subset $S \subseteq X$ of size $m \leq |S| \leq d$. Since the number of those subsets is $\sum_{i=m}^d \binom{k}{i}$, the number of vertices in \mathbf{K} is at most $k + \sum_{i=m}^d \binom{k}{i}$, as required for Item 2. For Item 3, notice that to encode the graph \mathbf{K} , it suffices to specify the adjacencies in $\mathbf{K}[X]$ and the existence of the vertex v_S in \mathbf{K} for each $S \subseteq X$ of size $m \leq |S| \leq d$, hence \mathbf{K} can be encoded in $\binom{k}{2} + \sum_{i=m}^d \binom{k}{i}$ bits.

We turn to the proof of Item 4. Let \mathbb{F} be a field with $m \leq m(\mathbb{F}, d)$. Suppose first that $\bar{\xi}_{\mathbb{F}}(G) \leq d$, that is, there exists a d -dimensional orthogonal representation $(u_v)_{v \in V}$ of G over \mathbb{F} . We define a d -dimensional orthogonal representation of \mathbf{K} over \mathbb{F} as follows. First, we

8:10 Kernelization for Orthogonality Dimension

assign to each vertex $v \in X$ the vector u_v . It clearly holds that every two vertices of X that are adjacent in K are assigned orthogonal vectors. Next, for each vertex v_S of K with $S \subseteq X$ and $m \leq |S| \leq d$, there exists a vertex $v \in V \setminus X$ such that $S \subseteq N_G(v)$. We assign to v_S the vector u_v of such a vertex v . Notice that such a vector is orthogonal to all the vectors associated with the vertices of S , i.e., the neighbors of v_S in K . This gives us a d -dimensional orthogonal representation of K over \mathbb{F} , implying that $\bar{\xi}_{\mathbb{F}}(K) \leq d$.

For the other direction, suppose that $\bar{\xi}_{\mathbb{F}}(K) \leq d$. Letting V' denote the vertex set of K , there exists a d -dimensional orthogonal representation $(u_v)_{v \in V'}$ of K over \mathbb{F} . We define a d -dimensional orthogonal representation of G over \mathbb{F} as follows. First, we assign to each vertex $v \in X$ the vector u_v . It clearly holds that every two vertices of X that are adjacent in G are assigned orthogonal vectors. We next extend this assignment to the vertices of the independent set $V \setminus X$ of G . Consider some vertex $v \in V \setminus X$, let $W = \text{span}(\{u_{v'} \mid v' \in N_G(v)\})$, and notice that v may be assigned any non-self-orthogonal vector of \mathbb{F}^d that lies in W^\perp . If $\dim(W) < m$, then by $m \leq m(\mathbb{F}, d)$, there exists a non-self-orthogonal vector in W^\perp , which can be assigned to the vertex v . Otherwise, there exists a set of vertices $S \subseteq N_G(v)$ of size $m \leq |S| \leq d$ whose vectors form a basis of W , that is, $W = \text{span}(\{u_{v'} \mid v' \in S\})$. By the definition of the graph K , it includes the vertex v_S , and its vector is orthogonal to the vectors $u_{v'}$ with $v' \in S$, and thus lies in W^\perp . This yields the existence of the desired vector for v , so we are done. \blacktriangleleft

With Lemma 16 at hand, we are ready to prove Theorem 14.

Proof of Theorem 14. The input of the $\text{ORTHO-DIM}_{\mathbb{F}}$ problem on $\text{EMPTY} + kv$ graphs consists of a graph G , a vertex cover X of G of size $|X| = k$, and an integer d . Consider the algorithm that given such an input acts as follows. If $d > k$ then the algorithm accepts. Otherwise, the algorithm calls an algorithm for the $\text{ORTHO-DIM}_{\mathbb{F}}$ problem on the input (K, d) , where $K = K(G, X, 1, d)$ is the graph given in Definition 15, and returns its answer. Note that we use here the assumption that the $\text{ORTHO-DIM}_{\mathbb{F}}$ problem is decidable.

For correctness, observe first that the input graph G is $(k + 1)$ -colorable, as follows by assigning k distinct colors to the vertices of the vertex cover X and another color to the vertices of the independent set $V \setminus X$. It thus follows, using Claim 12, that $\bar{\xi}_{\mathbb{F}}(G) \leq \chi(G) \leq k + 1$. Therefore, if $d > k$, then it holds that $\bar{\xi}_{\mathbb{F}}(G) \leq d$, hence our algorithm correctly accepts. Otherwise, the algorithm calls an algorithm for $\text{ORTHO-DIM}_{\mathbb{F}}$ on the input (K, d) . The correctness of its answer follows from Item 4 of Lemma 16, which guarantees that $\bar{\xi}_{\mathbb{F}}(G) \leq d$ if and only if $\bar{\xi}_{\mathbb{F}}(K) \leq d$.

We finally analyze the running time of the algorithm. On instances with $d > k$, the algorithm is clearly efficient. For instances with $d \leq k$, by Item 2 of Lemma 16, the number of vertices in K is $O(k^d) \leq O(k^k)$. Using the decidability of $\text{ORTHO-DIM}_{\mathbb{F}}$, this implies that the running time of the algorithm is bounded by $f(k) \cdot n^{O(1)}$ for some computable function f , where n stands for the input size. Therefore, the $\text{ORTHO-DIM}_{\mathbb{F}}$ problem on $\text{EMPTY} + kv$ graphs is fixed-parameter tractable. \blacktriangleleft

In order to apply Theorem 14 to the real field \mathbb{R} , one has to show that the $\text{ORTHO-DIM}_{\mathbb{R}}$ problem is decidable. We obtain this result using the problem of the existential theory of the reals, in which the input is a collection of equalities and inequalities of polynomials over the reals, and the goal is to decide whether there exists an assignment of real values to the variables satisfying all the constraints. In 1951, Tarski [31] proved that the problem is decidable. His result was strengthened in 1988 by Canny [10], who proved that it actually lies in the complexity class PSPACE . We derive the following simple consequence.

► **Proposition 17.** *The $\text{ORTHO-DIM}_{\mathbb{R}}$ problem lies in PSPACE.*

Proof. It is sufficient to show that the $\text{ORTHO-DIM}_{\mathbb{R}}$ problem is reducible in polynomial time to the problem of the existential theory of the reals, which lies in PSPACE [10]. Consider the reduction that given a graph $G = (V, E)$ and an integer d produces a collection P_G of polynomial constraints over the reals defined as follows. For each vertex $v \in V$, let $x_{v,1}, \dots, x_{v,d}$ denote d variables associated with v . For each vertex $v \in V$, add to P_G the inequality $\sum_{i=1}^d x_{v,i}^2 \neq 0$, and for each edge $\{v, v'\} \in E$, add to P_G the equality $\sum_{i=1}^d x_{v,i} \cdot x_{v',i} = 0$. The reduction returns the collection P_G , which can clearly be computed in polynomial time. Observe that $\bar{\xi}_{\mathbb{R}}(G) \leq d$ if and only if there exists an assignment over the reals satisfying the constraints of P_G , implying the correctness of the reduction. ◀

Proposition 17 implies that the $\text{ORTHO-DIM}_{\mathbb{R}}$ problem is decidable. Using Theorem 14, we obtain the following corollary, which combined with Theorem 13, confirms Theorem 2.

► **Corollary 18.** *The $\text{ORTHO-DIM}_{\mathbb{R}}$ problem on $\text{EMPTY} + kv$ graphs is fixed-parameter tractable.*

4 Kernelization for d -Ortho-Dim $_{\mathbb{F}}$ Parameterized by Vertex Cover

We consider now the d - $\text{ORTHO-DIM}_{\mathbb{F}}$ problem for a fixed constant d and study its kernelizability when parameterized by the vertex cover number (recall Definition 1). We first leverage our discussion from the previous section to derive Theorem 3, namely, to show that for every field \mathbb{F} and for every integer $d \geq 3$, the d - $\text{ORTHO-DIM}_{\mathbb{F}}$ problem on $\text{EMPTY} + kv$ graphs admits a kernel with $O(k^d)$ vertices and bit-size $O(k^d)$.

Proof of Theorem 3. Fix a field \mathbb{F} and an integer $d \geq 3$. The input of d - $\text{ORTHO-DIM}_{\mathbb{F}}$ on $\text{EMPTY} + kv$ graphs consists of a graph G and a vertex cover X of G of size $|X| = k$. Consider the algorithm that given such an input returns the pair (K, X) , where $K = K(G, X, 1, d)$ is the graph from Definition 15. Since d is a fixed constant, the algorithm can be implemented in polynomial time. By Lemma 16, the set X forms a vertex cover of K , the graph K has $O(k^d)$ vertices and bit-size $O(k^d)$, and the instances (G, X) and (K, X) are equivalent. This completes the proof. ◀

For the real field \mathbb{R} , we prove Theorem 5, which improves on the kernel provided by Theorem 3 to $O(k^{d-1})$ vertices and bit-size $O(k^{d-1} \cdot \log k)$. We start with a couple of auxiliary lemmas.

► **Lemma 19.** *For every integer d , if a graph has a d -dimensional orthogonal representation over \mathbb{R} , then it has a d -dimensional orthogonal representation over \mathbb{R} , all of whose vectors have 1 as their first entry.*

Proof. The proof applies the probabilistic method. Let d be an integer, let $G = (V, E)$ be a graph, and set $n = |V|$. Suppose that there exists a d -dimensional orthogonal representation $(u_v)_{v \in V}$ of G over \mathbb{R} . Let $a \in [2n]^d$ be a random d -dimensional vector, such that each entry of a is chosen from $[2n]$ uniformly at random. We observe that for every fixed nonzero vector $u \in \mathbb{R}^d$, it holds that $\langle a, u \rangle = 0$ with probability at most $\frac{1}{2n}$. Indeed, letting $i \in [d]$ be an index with $u_i \neq 0$, for every fixed choice of the values of a_j with $j \in [d] \setminus \{i\}$, there is at most one value of a_i in $[2n]$ for which it holds that $\langle a, u \rangle = 0$. By the union bound, it follows that the probability that there exists a vertex $v \in V$ such that $\langle a, u_v \rangle = 0$ is at most $n \cdot \frac{1}{2n} = \frac{1}{2}$. In particular, there exists a vector $a \in [2n]^d$ satisfying $\langle a, u_v \rangle \neq 0$ for all $v \in V$. Let us fix such a vector a .

8:12 Kernelization for Orthogonality Dimension

Now, let $M \in \mathbb{R}^{d \times d}$ be some orthonormal matrix (i.e., a matrix satisfying $M \cdot M^t = I_d$) whose first row is the vector a scaled to have Euclidean norm 1, i.e., $a/\|a\|$. We assign to each vertex $v \in V$ of the graph G the vector $M \cdot u_v$. Since M is orthonormal, it preserves inner products, hence this assignment forms a d -dimensional orthogonal representation of G over \mathbb{R} . Additionally, for every vertex $v \in V$, the first entry of the vector $M \cdot u_v$ is nonzero, because $\langle a, u_v \rangle \neq 0$. By scaling, one can obtain a d -dimensional orthogonal representation of G over \mathbb{R} , all of whose vectors have 1 as their first entry, as required. ◀

Before we state the next lemma, we need a brief preparation. For a field \mathbb{F} , a polynomial in $\mathbb{F}[x_1, \dots, x_n]$ is called homogeneous of degree d if each of its monomials has degree d . Note that the zero polynomial is homogeneous of degree d for every $d \geq 0$. A monomial is called multilinear if it forms a product of distinct variables, and a polynomial is called multilinear if it forms a linear combination of multilinear monomials. For example, the determinant of $d \times d$ matrices over a field \mathbb{F} , viewed as a polynomial on d^2 variables, is multilinear and homogeneous of degree d . Moreover, it is a linear combination of $d!$ monomials, each of which forms a product of d variables, one taken from each row of the matrix. Note that the dimension over \mathbb{F} of the vector space of multilinear homogeneous polynomials of degree d in $\mathbb{F}[x_1, \dots, x_n]$ is $\binom{n}{d}$.

► **Lemma 20.** *For every integer d , there exists a multilinear homogeneous polynomial $p : \mathbb{R}^{d \times d} \rightarrow \mathbb{R}$ of degree $d - 1$, defined on d^2 variables corresponding to the entries of a $d \times d$ matrix, such that for every matrix $M \in \mathbb{R}^{d \times d}$ whose first row is the all-one vector, it holds that $p(M) = 0$ if and only if there exists a nonzero vector in \mathbb{R}^d that is orthogonal to all columns of M .*

Proof. For an integer d , consider the determinant polynomial $\det : \mathbb{R}^{d \times d} \rightarrow \mathbb{R}$. It is well known that for every matrix $M \in \mathbb{R}^{d \times d}$, it holds that $\det(M) = 0$ if and only if the columns of M span a subspace of dimension smaller than d , and that this condition is equivalent to the existence of a nonzero vector in \mathbb{R}^d that is orthogonal to all columns of M . Recall that \det is a multilinear polynomial, with each monomial being a product of d variables, each selected from a different row of the matrix. Let $p : \mathbb{R}^{d \times d} \rightarrow \mathbb{R}$ be the polynomial obtained from \det by substituting 1 for the variables that correspond to the first row of the matrix, and observe that p is a multilinear homogeneous polynomial of degree $d - 1$. Note that although p is defined on d^2 variables, it actually depends on only $d^2 - d$ of them. We finally observe that for every matrix $M \in \mathbb{R}^{d \times d}$ whose first row is the all-one vector, it holds that $p(M) = 0$ if and only if there exists a nonzero vector in \mathbb{R}^d that is orthogonal to all columns of M . This completes the proof. ◀

We are ready to prove Theorem 5, providing a kernel with $O(k^{d-1})$ vertices and bit-size $O(k^{d-1} \cdot \log k)$ for the d -ORTHO-DIM $_{\mathbb{R}}$ problem on EMPTY + kv graphs for all integers $d \geq 3$.

Proof of Theorem 5. Fix an integer $d \geq 3$. The input of d -ORTHO-DIM $_{\mathbb{R}}$ on EMPTY + kv graphs consists of a graph $G = (V, E)$ and a vertex cover $X \subseteq V$ of G of size $|X| = k$. Consider the algorithm that given such an input acts in two phases, as described next.

In the first phase, the algorithm constructs the graph $G' = \mathbf{K}(G, X, d, d)$ given in Definition 15. Let V' denote the vertex set of G' , and recall that every vertex $v_S \in V' \setminus X$ is associated with some set $S \subseteq X$ of size $|S| = d$ such that $N_{G'}(v_S) = S$. By Lemma 16, the set X is a vertex cover of G' , and it holds that $|V'| \leq k + \binom{k}{d}$.

In the second phase, the algorithm constructs a graph G'' . To do so, the algorithm first associates with each vertex $v \in X$ a d -dimensional vector x_v of variables over \mathbb{R} . Note that the total number of variables is $k \cdot d$. For each vertex $v_S \in V' \setminus X$, we apply Lemma 20 to

obtain a multilinear homogeneous polynomial p_S of degree $d - 1$, defined on the d^2 variables associated with the d neighbors of v_S in G' (which all lie in X). The polynomial p_S satisfies that for every assignment $M \in \mathbb{R}^{d \times d}$ to its variables with first row equal to the all-one vector, it holds that $p_S(M) = 0$ if and only if there exists a nonzero vector in \mathbb{R}^d that is orthogonal to all columns of M . Let $P = \text{span}(\{p_S \mid v_S \in V' \setminus X\})$ denote the subspace spanned by the polynomials associated with the vertices of $V' \setminus X$. The algorithm proceeds by finding a set $Y \subseteq V' \setminus X$, such that the polynomials associated with the vertices of Y form a basis for P . Note that P is contained in the vector space of multilinear homogeneous polynomials of degree $d - 1$ on $k \cdot d$ variables. Since the dimension of the latter is $\binom{k \cdot d}{d-1}$, recalling that d is a fixed constant, it follows that $|Y| \leq \binom{k \cdot d}{d-1} \leq O(k^{d-1})$. Letting $V'' = X \cup Y$, the algorithm returns the graph $G'' = G'[V'']$ and the set X , which forms a vertex cover of G'' because it forms a vertex cover of G' .

The number of vertices in G'' is $|V''| = |X| + |Y| \leq k + O(k^{d-1}) = O(k^{d-1})$. The number of edges in $G''[X]$ is at most $\binom{k}{2}$, and since the degree of each vertex of Y is d , the number of edges in G'' that involve vertices of Y is $d \cdot |Y|$. It follows that the total number of edges in G'' is at most $\binom{k}{2} + d \cdot O(k^{d-1}) \leq O(k^{d-1})$. Therefore, the number of bits required to encode the edges of G'' is at most $O(k^{d-1} \cdot \log |V''|) \leq O(k^{d-1} \cdot \log k)$, as required.

It is not difficult to verify that the algorithm can be implemented in polynomial time. Note that the set Y can be calculated in polynomial time by applying Gaussian elimination with $\binom{k \cdot d}{d-1}$ variables.

For the correctness of the algorithm, we shall prove that $\bar{\xi}_{\mathbb{R}}(G) \leq d$ if and only if $\bar{\xi}_{\mathbb{R}}(G'') \leq d$. By Item 4 of Lemma 16, using $m(\mathbb{R}, d) = d$ (see Remark 9), it holds that $\bar{\xi}_{\mathbb{R}}(G) \leq d$ if and only if $\bar{\xi}_{\mathbb{R}}(G') \leq d$. It thus suffices to show that $\bar{\xi}_{\mathbb{R}}(G') \leq d$ if and only if $\bar{\xi}_{\mathbb{R}}(G'') \leq d$.

It obviously holds that if $\bar{\xi}_{\mathbb{R}}(G') \leq d$ then $\bar{\xi}_{\mathbb{R}}(G'') \leq d$, because G' contains G'' as a subgraph. For the converse, suppose that $\bar{\xi}_{\mathbb{R}}(G'') \leq d$, that is, there exists a d -dimensional orthogonal representation of G'' over \mathbb{R} . By Lemma 19, it further follows that there exists a d -dimensional orthogonal representation $(u_v)_{v \in V''}$ of G'' over \mathbb{R} , such that every vector u_v has 1 as its first entry. For each vertex $v \in X$, assign the vector u_v to the vertex v as well as to the variables of the vector x_v associated with v . We will show that this assignment to the vertices of X can be extended to an orthogonal representation of G' over \mathbb{R} . Indeed, for every vertex $v_S \in Y$ of G'' , the nonzero vector u_{v_S} is orthogonal to the vectors of the vertices of S . This implies, using Lemma 20 and the fact that the first entries of the vectors x_v with $v \in X$ are all 1, that the polynomial p_S vanishes on this assignment. Since the polynomials p_S with $v_S \in Y$ form a basis of the subspace P , it follows that all the polynomials p_S associated with the vertices $v_S \in V' \setminus X$ vanish on this assignment as well. Using Lemma 20 again, we obtain that for each vertex $v_S \in V' \setminus X$, there exists a nonzero vector that is orthogonal to the vectors of the vertices of S , and these are precisely the neighbors of v_S in G' . This gives us a d -dimensional orthogonal representation of G' over \mathbb{R} , which yields that $\bar{\xi}_{\mathbb{R}}(G') \leq d$, concluding the proof. \blacktriangleleft

5 Lower Bound

In this section, we prove our lower bound on the kernel size of the d -ORTHO-DIM $_{\mathbb{F}}$ problem parameterized by the vertex cover number. We first present the gadget graph that will be used in the proof.

5.1 Gadget Graph

A key ingredient in the proof of our lower bound is the following lemma, which generalizes a construction of [29] (corresponding to the case of $d = 3$). Here, two nonzero vectors $u_1, u_2 \in \mathbb{F}^d$ are said to be proportional if there exists some $\alpha \in \mathbb{F}$ such that $u_1 = \alpha \cdot u_2$. The proof can be found in the full version of the paper.

► **Lemma 21.** *For an integer $d \geq 3$, let C_{2d} denote the cycle graph on $2d$ vertices, let x_0 and x_1 denote two adjacent vertices in the cycle, and let $H = \overline{C_{2d}}$ denote its complement graph.*

1. *There exists a proper d -coloring of H that assigns to x_0 and x_1 the same color.*
2. *There exists a proper d -coloring of H that assigns to x_0 and x_1 distinct colors.*
3. *For every field \mathbb{F} and for every d -dimensional orthogonal representation of H over \mathbb{F} , the vectors assigned to x_0 and x_1 are either orthogonal or proportional.*

5.2 The d -Ortho-Dim $_{\mathbb{F}}$ Problem on Empty + kv Graphs

We prove the following lower bound on the size of any compression for the d -ORTHO-DIM $_{\mathbb{F}}$ problem parameterized by the vertex cover number.

► **Theorem 22.** *For every field \mathbb{F} , every integer $d \geq 3$, and any real $\varepsilon > 0$, the d -ORTHO-DIM $_{\mathbb{F}}$ problem on EMPTY + kv graphs does not admit a compression with bit-size $O(k^{d-1-\varepsilon})$ unless $\text{NP} \subseteq \text{coNP/poly}$.*

Note that Theorem 22 confirms Theorem 4. Another immediate corollary is the following.

► **Corollary 23.** *For every field \mathbb{F} , the ORTHO-DIM $_{\mathbb{F}}$ problem on EMPTY + kv graphs does not admit a polynomial compression unless $\text{NP} \subseteq \text{coNP/poly}$.*

The starting point of the proof of Theorem 22 is the following theorem, which summarizes the lower bounds proved in [23, 24] on the size of any compression for the q -COLORING problem parameterized by the vertex cover number (see [24, Corollary 2]).

► **Theorem 24** ([23, 24]). *For every integer $q \geq 3$, the q -COLORING problem on EMPTY + kv graphs does not admit a compression with bit-size $O(k^{q-1-\varepsilon})$ unless $\text{NP} \subseteq \text{coNP/poly}$.*

Equipped with Lemma 21, we relate the d -COLORING and d -ORTHO-DIM $_{\mathbb{F}}$ problems parameterized by the vertex cover number, as stated below.

► **Theorem 25.** *For every field \mathbb{F} and for every integer $d \geq 3$, there exists a linear-parameter transformation from d -COLORING on EMPTY + kv graphs to d -ORTHO-DIM $_{\mathbb{F}}$ on EMPTY + kv graphs.*

Proof. Fix a field \mathbb{F} and an integer $d \geq 3$. Consider an instance of the d -COLORING problem on EMPTY + kv graphs, namely, a graph $G = (V, E)$ and a vertex cover $X \subseteq V$ of G of size $|X| = k$. Our goal is to construct in polynomial time a graph $G' = (V', E')$ and a vertex cover $X' \subseteq V'$ of G' of size $|X'| = O(k)$, such that $\chi(G) \leq d$ if and only if $\bar{\xi}_{\mathbb{F}}(G') \leq d$.

To do so, we start with the graph G and add to it a clique of size d whose vertices are denoted by z_1, \dots, z_d . Then, for each index $i \in [d]$ and each vertex $v \in X$, we add to the graph a copy $H_{i,v}$ of the complement $\overline{C_{2d}}$ of the cycle graph on $2d$ vertices, where two consecutive vertices of the cycle are identified with the vertices z_i and v . Note that we add here $d \cdot k$ such gadgets to the graph and that each of them involves $2d - 2$ new vertices. Let $G' = (V', E')$ denote the obtained graph, and let X' denote the set that consists of the

vertices of X and the vertices that were added to G in the construction. The transformation returns the pair (G', X') . Since X is a vertex cover of G , the set $V \setminus X$ is an independent set of G . It follows that $V' \setminus X'$ is an independent set of G' , hence X' is a vertex cover of G' . Its size satisfies $|X'| = k + d + d \cdot k \cdot (2d - 2) = O(k)$, hence the transformation is linear-parameter. The transformation can clearly be implemented in polynomial time. For correctness, we shall prove that $\chi(G) \leq d$ if and only if $\bar{\xi}_{\mathbb{F}}(G') \leq d$.

Suppose first that $\chi(G) \leq d$, and consider some proper d -coloring of G with color set $[d]$. We extend this coloring to a d -coloring of G' as follows. First, for each $i \in [d]$, we assign the color i to the vertex z_i . Clearly, no edge that connects two of the vertices z_1, \dots, z_d is monochromatic. Next, for each $i \in [d]$ and $v \in X$, consider the vertices of the component $H_{i,v}$. The only vertices of $H_{i,v}$ that already received colors are z_i and v . By Lemma 21, this partial coloring of $H_{i,v}$ can be extended to a proper d -coloring of the whole gadget. Indeed, if z_i and v are assigned the same color then this follows from Item 1 of the lemma, and if z_i and v are assigned distinct colors then this follows from Item 2 of the lemma. This gives us a proper d -coloring of G' , which implies using Claim 12 that $\bar{\xi}_{\mathbb{F}}(G') \leq \chi(G') \leq d$.

For the converse direction, suppose that $\bar{\xi}_{\mathbb{F}}(G') \leq d$, and consider a d -dimensional orthogonal representation $(u_v)_{v \in V'}$ of G' over \mathbb{F} . Since the vertices z_1, \dots, z_d form a clique in G' , it follows that their vectors u_{z_1}, \dots, u_{z_d} are pairwise orthogonal. Since they are non-self-orthogonal, it follows that they are linearly independent, and thus span the entire vector space \mathbb{F}^d . For each $i \in [d]$ and $v \in X$, consider the vectors assigned by the given orthogonal representation to the vertices of the component $H_{i,v}$ in G' , and apply Item 3 of Lemma 21 to obtain that the vectors u_{z_i} and u_v are either orthogonal or proportional. However, the vectors u_{z_1}, \dots, u_{z_d} span the vector space \mathbb{F}^d , hence for each $v \in X$, the nonzero vector u_v cannot be orthogonal to all of them. This yields that for each vertex $v \in X$, the vector u_v is proportional to exactly one of the vectors u_{z_1}, \dots, u_{z_d} .

We define a d -coloring of G as follows. To each vertex $v \in X$, assign the color $i \in [d]$ for which u_v is proportional to u_{z_i} . Since the given orthogonal representation assigns orthogonal vectors to adjacent vertices, it follows that this coloring assigns distinct colors to adjacent vertices in X . Next, to each vertex $v \in V \setminus X$, assign a color from $[d]$ that does not appear on its neighbors. Notice that all the neighbors of v lie in X and were already colored, because X is a vertex cover of G . To see that such a color exists, recall that the vector u_v is nonzero and orthogonal to the vectors associated with its neighbors in X by the given orthogonal representation of G' . Since every such vector is proportional to one of u_{z_1}, \dots, u_{z_d} , it follows that there exists some $i \in [d]$ for which no neighbor of v is associated with a vector proportional to u_{z_i} , yielding the existence of the desired color for v . This gives us a proper d -coloring of G and implies that $\chi(G) \leq d$, so we are done. ◀

We finally combine Theorems 24 and 25 to derive Theorem 22.



Proof of Theorem 22. Fix a field \mathbb{F} , an integer $d \geq 3$, and a real $\varepsilon > 0$. By Theorem 25, there exists a linear-parameter transformation from d -COLORING on EMPTY + kv graphs to d -ORTHO-DIM $_{\mathbb{F}}$ on EMPTY + kv graphs. Therefore, if d -ORTHO-DIM $_{\mathbb{F}}$ on EMPTY + kv graphs admits a compression with bit-size $O(k^{d-1-\varepsilon})$, then by composing this compression with the given transformation, it follows that d -COLORING on EMPTY + kv graphs admits a compression with bit-size $O(k^{d-1-\varepsilon})$ as well. By Theorem 24, this implies that $\text{NP} \subseteq \text{coNP/poly}$, and we are done. ◀

References

- 1 Ziv Bar-Yossef, Yitzhak Birk, T. S. Jayram, and Tomer Kol. Index coding with side information. *IEEE Trans. Inform. Theory*, 57(3):1479–1494, 2011. Preliminary version in FOCS’06. doi:10.1109/TIT.2010.2103753.
- 2 Alexander Barg and Gilles Zémor. High-rate storage codes on triangle-free graphs. *IEEE Trans. Inform. Theory*, 68(12):7787–7797, 2022. doi:10.1109/TIT.2022.3191309.
- 3 Yitzhak Birk and Tomer Kol. Coding on demand by an informed source (ISCOD) for efficient broadcast of different supplemental data to caching clients. *IEEE Trans. Inform. Theory*, 52(6):2825–2830, 2006. Preliminary version in INFOCOM’98. doi:10.1109/TIT.2006.874540.
- 4 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009. Preliminary version in ICALP’08. doi:10.1016/J.JCSS.2009.04.001.
- 5 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM J. Discret. Math.*, 28(1):277–305, 2014. Preliminary version in STACS’11. doi:10.1137/120880240.
- 6 Jop Briët, Harry Buhrman, Debbie Leung, Teresa Piovesan, and Florian Speelman. Round elimination in exact communication complexity. In *Proc. of the 10th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC’15)*, pages 206–225, 2015. doi:10.4230/LIPICS.TQC.2015.206.
- 7 Jop Briët and Jeroen Zuiddam. On the orthogonal rank of Cayley graphs and impossibility of quantum round elimination. *Quantum Information & Computation*, 17(1&2):106–116, 2017.
- 8 Leizhen Cai. Parameterized complexity of vertex colouring. *Discret. Appl. Math.*, 127(3):415–429, 2003. doi:10.1016/S0166-218X(02)00242-1.
- 9 Peter J. Cameron, Ashley Montanaro, Michael W. Newman, Simone Severini, and Andreas J. Winter. On the quantum chromatic number of a graph. *Electron. J. Comb.*, 14(1):R81, 2007. doi:10.37236/999.
- 10 John F. Canny. Some algebraic and geometric computations in PSPACE. In *Proc. of the 20th Annual ACM Symposium on Theory of Computing (STOC’88)*, pages 460–467, 1988. doi:10.1145/62212.62257.
- 11 Dror Chawin and Ishay Haviv. On the subspace choosability in graphs. *Electron. J. Comb.*, 29(2):P2.20, 2022. Preliminary version in EuroComb’21. doi:10.37236/10765.
- 12 Dror Chawin and Ishay Haviv. Improved NP-hardness of approximation for orthogonality dimension and minrank. *SIAM J. Discret. Math.*, 37(4):2670–2688, 2023. Preliminary version in STACS’23. doi:10.1137/23M155760X.
- 13 Bruno Codenotti, Pavel Pudlák, and Giovanni Resta. Some structural properties of low-rank matrices related to computational complexity. *Theor. Comput. Sci.*, 235(1):89–107, 2000. doi:10.1016/S0304-3975(99)00185-1.
- 14 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
- 15 Ronald de Wolf. Quantum Computing and Communication Complexity. Ph.D. Thesis, Universiteit van Amsterdam, 2001.
- 16 Fedor V. Fomin, Bart M. P. Jansen, and Michal Pilipczuk. Preprocessing subgraph and minor problems: When does a small vertex cover help? *J. Comput. Syst. Sci.*, 80(2):468–495, 2014. Preliminary version in IPEC’12. doi:10.1016/J.JCSS.2013.09.004.
- 17 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019.
- 18 Alexander Golovnev and Ishay Haviv. The (generalized) orthogonality dimension of (generalized) Kneser graphs: Bounds and applications. *Theory of Computing*, 18(22):1–22, 2022. Preliminary version in CCC’21. URL: <https://theoryofcomputing.org/articles/v018a022/>.

- 19 Alexander Golovnev, Oded Regev, and Omri Weinstein. The minrank of random graphs. *IEEE Trans. Inform. Theory*, 64(11):6990–6995, 2018. Preliminary version in RANDOM’17. doi:10.1109/TIT.2018.2810384.
- 20 Ishay Haviv. Approximating the orthogonality dimension of graphs and hypergraphs. *Chic. J. Theor. Comput. Sci.*, 2022(2):1–26, 2022. Preliminary version in MFCS’19.
- 21 Gerald Haynes, Catherine Park, Amanda Schaeffer, Jordan Webster, and Lon H. Mitchell. Orthogonal vector coloring. *Electron. J. Comb.*, 17(1):R55, 2010. doi:10.37236/327.
- 22 Bart M. P. Jansen. The Power of Data Reduction: Kernels for Fundamental Graph Problems. Ph.D. Thesis, Utrecht University, 2013.
- 23 Bart M. P. Jansen and Stefan Kratsch. Data reduction for graph coloring problems. *Inf. Comput.*, 231:70–88, 2013. Preliminary version in FCT’11. doi:10.1016/J.IC.2013.08.005.
- 24 Bart M. P. Jansen and Astrid Pieterse. Optimal data reduction for graph coloring using low-degree polynomials. *Algorithmica*, 81(10):3865–3889, 2019. Preliminary version in IPEC’17. doi:10.1007/S00453-019-00578-5.
- 25 Bart M. P. Jansen and Astrid Pieterse. Optimal sparsification for some binary CSPs using low-degree polynomials. *ACM Trans. Comput. Theory*, 11(4):28:1–26, 2019. Preliminary version in MFCS’16. doi:10.1145/3349618.
- 26 László Lovász. On the Shannon capacity of a graph. *IEEE Trans. Inform. Theory*, 25(1):1–7, 1979. doi:10.1109/TIT.1979.1055985.
- 27 László Lovász. *Graphs and Geometry*, volume 65. Colloquium Publications, 2019.
- 28 László Lovász, Michael Saks, and Alexander Schrijver. Orthogonal representations and connectivity of graphs. *Linear Algebra Appl.*, 114–115:439–454, 1989. Special Issue Dedicated to Alan J. Hoffman.
- 29 René Peeters. Orthogonal representations over finite fields and the chromatic number of graphs. *Combinatorica*, 16(3):417–431, 1996. doi:10.1007/BF01261326.
- 30 Søren Riis. Information flows, graphs and their guessing numbers. *Electron. J. Comb.*, 14(1):R44, 2007. Preliminary version in NETCOD’06. doi:10.37236/962.
- 31 Alfred Tarski. *A Decision Method for Elementary Algebra and Geometry*. Number R-109 in RAND Research & Commentary Reports. RAND, Santa Monica, CA, 1951.
- 32 Chee-Keng Yap. Some consequences of non-uniform conditions on uniform classes. *Theor. Comput. Sci.*, 26(3):287–300, 1983. doi:10.1016/0304-3975(83)90020-8.

Fine-Grained Complexity of Multiple Domination and Dominating Patterns in Sparse Graphs

Marvin Künnemann  

Karlsruhe Institute of Technology, Germany

Mirza Redzic  

Karlsruhe Institute of Technology, Germany

Abstract

The study of domination in graphs has led to a variety of *dominating set* problems studied in the literature. Most of these follow the following general framework: Given a graph G and an integer k , decide if there is a set S of k vertices such that (1) some inner connectivity property $\phi(S)$ (e.g., connectedness) is satisfied, and (2) each vertex v satisfies some domination property $\rho(S, v)$ (e.g., there is some $s \in S$ that is adjacent to v).

Since many real-world graphs are *sparse*, we seek to determine the optimal running time of such problems in both the number n of vertices and the number m of edges in G . While the classic dominating set problem admits a rather limited improvement in sparse graphs (Fischer, Künnemann, Redzic SODA'24), we show that natural variants studied in the literature admit much larger speed-ups, with a diverse set of possible running times. Specifically, using fast matrix multiplication we devise efficient algorithms which in particular yield the following conditionally optimal running times if the matrix multiplication exponent ω is equal to 2:

- *r-Multiple k-Dominating Set* (each vertex v must be adjacent to at least r vertices in S): If $r \leq k - 2$, we obtain a running time of $(m/n)^r n^{k-r+o(1)}$ that is conditionally optimal assuming the 3-uniform hyperclique hypothesis. In sparse graphs, this fully interpolates between $n^{k-1 \pm o(1)}$ and $n^{2 \pm o(1)}$, depending on r . Curiously, when $r = k - 1$, we obtain a randomized algorithm beating $(m/n)^{k-1} n^{1+o(1)}$ and we show that this algorithm is close to optimal under the k -clique hypothesis.
- *H-Dominating Set* (S must induce a pattern H). We conditionally settle the complexity of three such problems: (a) Dominating Clique (H is a k -clique), (b) Maximal Independent Set of size k (H is an independent set on k vertices), (c) Dominating Induced Matching (H is a perfect matching on k vertices). For all sufficiently large k , we provide algorithms with running time $(m/n)m^{(k-1)/2+o(1)}$ for (a) and (b), and $m^{k/2+o(1)}$ for (c). We show that these algorithms are essentially optimal under the k -Orthogonal Vectors Hypothesis (k -OVH). This is in contrast to H being the k -Star, which is susceptible only to a very limited improvement, with the best algorithm running in time $n^{k-1 \pm o(1)}$ in sparse graphs under k -OVH.

2012 ACM Subject Classification Theory of computation \rightarrow Graph algorithms analysis; Theory of computation \rightarrow Parameterized complexity and exact algorithms; Theory of computation \rightarrow Problems, reductions and completeness

Keywords and phrases Fine-grained complexity theory, Dominating set, Sparsity in graphs, Conditionally optimal algorithms

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.9

Related Version : <https://arxiv.org/abs/2409.08037>

Funding Research supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 462679611.



© Marvin Künnemann and Mirza Redzic;

licensed under Creative Commons License CC-BY 4.0

19th International Symposium on Parameterized and Exact Computation (IPEC 2024).

Editors: Édouard Bonnet and Paweł Rzażewski; Article No. 9; pp. 9:1–9:18

Leibniz International Proceedings in Informatics



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

1 Introduction

Domination in graphs is among the central topics in graph theory. Although the earliest evidence of interest in concepts related to domination can be traced back to the mid 1800s in connection with various chess problems, it was introduced only a century later, in 1958, as a graph-theoretical concept by Claude Berge. It has since gained a lot of attention and has been well-studied from both a graph-theoretic perspective, e.g., [2, 11, 19, 48, 49], and an algorithmic perspective, e.g., [22, 46, 30, 31, 37, 47]. This problem has also played a central role in the field of complexity theory. Besides being one of the classic NP-complete problems, the Dominating Set problem has proven valuable within the realm of parameterized complexity theory, where it is regarded as perhaps the most natural $W[2]$ -complete problem when parameterized by the solution size k [20], as well as fine-grained complexity in P, where the k -Dominating Set problem (for fixed k) was among the first problems for which tight lower bounds under the Strong Exponential Time Hypothesis (SETH) have been established [46].

Over the years, the concept of domination in graphs has spawned many natural variations, each offering unique insights into the structural properties of a graph, as well as different practical applications (e.g. in analysing sensor networks, facility management, studying influence in social networks, etc.). Some examples of such variations include total domination, paired domination, independent domination, multiple domination, etc. Most of these domination problems satisfy the following framework: We are given a graph G and an integer k and we want to decide if there exists a set of vertices $S = \{x_1, \dots, x_k\}$ that satisfies some fixed inner property $\phi(x_1, \dots, x_k)$ such that for every vertex $v \in V(G)$ the *domination* property $\rho(x_1, \dots, x_k, v)$ is satisfied. Some examples of inner properties ϕ include:

- x_1, \dots, x_k are connected (Connected Domination).
- x_1, \dots, x_k form an independent set (Independent Domination).
- Each $x_i \in S$ is adjacent to at least one $x_j \in S \setminus \{x_i\}$ (Total Domination).

Examples of the domination property ρ include:

- There exists $x_i \in S$ such that $d(x_i, v) \leq r$ (Domination at Distance r).
- v is adjacent to at least r distinct vertices $x_{i_1}, \dots, x_{i_r} \in S$ (r -Multiple Domination).
- There exists a path of length r between v and some $x_i \in S$ (r -Step Domination).

Many of these domination problems have not seen any polynomial improvements over brute force in dense graphs, i.e., the best known algorithms for finding a solution of size k typically run in $\Omega(n^k)$ time and for some variants it has been shown that improving upon these algorithms significantly would refute popular fine-grained complexity assumptions. Most notably, Pătraşcu and Williams [46] show that an $\mathcal{O}(n^{k-\epsilon})$ algorithm solving k -Dominating Set, for any $k \geq 3$ and $\epsilon > 0$, would refute the Strong Exponential Time Hypothesis (SETH). However, by far not all graphs of interest are dense. Particularly, many real-world graphs, for which the domination problems have been extensively used, are typically sparse (e.g. social networks, sensor networks, road networks, etc.). Hence, it is natural to ask what is the best running time of domination problems in sparse graphs. Recently, Fischer, Künnemann and Redzic [29] proved that the fine-grained complexity of k -Dominating set shows a non-trivial sensitivity to sparsity of the input graph. More precisely, despite the SETH-based lower bound of $n^{k-o(1)}$, they prove that when the input graph is sufficiently sparse, we can in fact improve upon this running time significantly by using sparse matrix multiplication techniques, and obtain a conditionally optimal running time of $mn^{k-2+o(1)}$ for all $k \geq 7$.¹ This raises the question if we can obtain similar improvements in sparse graphs for other

¹ In fact, this running time is achieved for all $k \geq 2$ if the matrix multiplication exponent ω is equal to 2.

natural domination problems. In this paper we consider two natural classes of domination problems that exhibit an interesting sensitivity to sparsity, namely *r-Multiple Domination* and *Pattern Domination*.

Multiple Domination in Graphs. The concept of Multiple Domination has been introduced as a generalization of Dominating Set by Fink and Jacobson in 1985 [27, 28] and has been intensively studied since (see e.g. [3, 4, 5, 6, 32, 36, 42]). For a graph $G = (V, E)$ we say a subset of vertices S is an *r-multiple dominating set* if each vertex $v \in V \setminus S$ has at least r neighbours in S .² Given a graph G with n vertices and m edges, the *r-Multiple k-Dominating Set* problem is to decide if there is an r -multiple dominating set S of size at most k . Harary and Haynes [34, 35] introduced, in two papers published in 1996 and 2000, a very related concept of double domination and, more generally, the *r-Tuple Dominating Set*, which is a subset of vertices S , such that the closed neighborhood of every vertex $v \in V$ intersects with S in at least r elements. We note that all of the algorithms and lower bounds that we provide for r -Multiple k -Dominating Set work with very minor modifications for r -Tuple Dominating Set as well.

We aim to settle the fine grained complexity of this problem in sparse graphs. Interestingly, the hardness of this problem depends not only on the trade-off between m and n , but also on the trade-off between r and k . In particular, we distinguish between the two cases, $r \leq k - 2$ and $r = k - 1$ (note that if $r \geq k$, the problem becomes trivial) – it turns out that the fine-grained complexity for these two cases differs already in dense graphs.

We note that most of the algorithms that we present in this paper use fast matrix multiplication. The resulting running times thus depend on the matrix multiplication exponent ω . Since resolving the precise value of ω is a fundamental open question in theoretical computer science, we aim to obtain upper bounds and conditional lower bounds that are as close as possible without knowing the true value of ω . To this end, we often focus on the case $\omega = 2$, for which our algorithms are usually tight (still, we include the precise running time of each algorithm in its respective technical section). In fact, in many cases we obtain matching upper and lower bounds already with the current bounds on ω , provided that k is sufficiently large.

Let us begin with a baseline algorithm for the case $r \leq k - 2$:

► **Theorem 1.** *Let $k \geq 3$ and $r \leq k - 2$ be fixed constants. Given a graph G with n vertices and m edges we can solve r -Multiple k -Dominating Set in time $\mathcal{O}\left(\left(\frac{m}{n}\right)^r n^{k-r+1}\right)$. Using fast matrix multiplication, this bound improves to $(m/n)^r n^{k-r+o(1)}$ if $\omega = 2$.*

Let us discuss the obtained upper bound, using for ease of presentation the case of $\omega = 2$: We note the remarkable improvement by a factor of $\Theta\left(\frac{n^{2r}}{m^r}\right)$ over the best known algorithm in dense graphs. Already for Double k -Dominating Set (i.e., $r = 2$), this yields an algorithm running in $m^2 n^{k-4+o(1)}$, which beats the running time of the best k -Dominating Set algorithm [29] by a factor of $\Theta(n^2/m)$. Even better, if we want each vertex in our graph to be dominated by precisely 50% of the solution vertices, we get an algorithm running in $m^{\frac{k}{2}+o(1)}$, halving the exponent in sparse graphs. Perhaps surprisingly, for r close to k , this yields an algorithm whose running time exponent is independent of k when the input graph is very sparse. In particular, for $r = k - 2$, this running time becomes essentially quadratic in very sparse

² We remark that in the literature, this concept is better known under the name *r-Dominating Set*. In the setting of parameterized complexity, however, the notion of k -Dominating Set usually refers to dominating sets of size k , so for clarity, we use the term *r-Multiple Dominating Set*.

graphs ($m = \tilde{O}(n)$). The question remains whether this running time is best possible – perhaps we can always obtain $m^{\frac{k}{2}+o(1)}$ (or even better) running time when $r \geq 2$? We answer this question negatively, and in fact show that any polynomial improvement over our algorithm (up to resolving the matrix multiplication exponent ω) would refute the 3-uniform Hyperclique Hypothesis³, thus settling the fine-grained complexity of this problem in sparse graphs whenever $r \leq k - 2$.

► **Theorem 2.** *Let $k \geq 3$, $r \leq k - 2$ be fixed constants, and $\varepsilon > 0$. An algorithm solving r -Multiple k -Dominating Set in time $\mathcal{O}\left(\left(\frac{m}{n}\right)^r n^{k-r-\varepsilon}\right)$ would refute the 3-uniform Hyperclique Hypothesis. This holds even when restricting $m = \Theta(n^{1+\gamma})$ for any $0 < \gamma \leq 1$.*

While the algorithmic approach of Theorem 1 is applicable also for the remaining case of $r = k - 1$, it turns out that the resulting upper bound of $(m/n)^{k-1} n^{1+o(1)}$ (if $\omega = 2$), is not optimal in general: In fact, we reduce the problem to Clique Detection, by observing that each pair of solution vertices dominates the whole graph (i.e., forms a dominating set of size 2). The resulting algorithm substantially improves over exhaustive search already in dense graphs. Furthermore, in sparse graphs, we can apply a Bloom-filter inspired randomized algorithm of [29], allowing us to list all 2-dominating sets efficiently, to obtain an efficient randomized reduction to an Unbalanced k -Clique Detection instance with $k - 1$ parts of size $\mathcal{O}\left(\frac{m}{n}\right)$ and one of size n , which we denote as k -Clique $\left(\frac{m}{n}, \dots, \frac{m}{n}, n\right)$.⁴

► **Theorem 3.** *For any fixed constant $k \geq 2$, let $T_k(m, n)$ denote the time required to solve the k -Clique $\left(\frac{m}{n}, \dots, \frac{m}{n}, n\right)$ problem. There is a randomized algorithm solving $(k - 1)$ -Multiple k -Dominating Set in time $m^{\frac{\omega}{2}+o(1)} + \mathcal{O}(T_k(m, n))$.*

It remains to analyze the complexity of the Unbalanced Clique problem. It is straightforward to obtain an algorithm solving this problem in time $\frac{n^2}{m} \cdot \left(\frac{m}{n}\right)^{\omega k/3+o(1)}$, which in case of very sparse graphs ($m = \tilde{O}(n)$) yields a near-linear running time. However, this running time analysis is still crude, and we can do even better for sufficiently “nice” values of m/n and k . More precisely, we show that for each positive integer p , we can solve k -Clique $\left(n^{\frac{1}{p}}, \dots, n^{\frac{1}{p}}, n\right)$ in time $\left(n^{\frac{1}{p}(k-1)+1}\right)^{\frac{\omega}{3}+o(1)}$ for all sufficiently large k satisfying $k \equiv 2p + 1 \pmod{3}$.

► **Proposition 4.** *Let $G = (V_1, \dots, V_k, E)$ be a k -partite graph with vertex part sizes $|V_1| = n$ and $|V_2| = \dots = |V_k| = n^\gamma$ for some $0 \leq \gamma \leq 1$. If*

1. $(k - 1 + \frac{1}{\gamma})$ is an integer divisible by 3,
2. $\frac{2}{\gamma} < k - 1$,

then we can decide if G has a clique of size k in time $(n^{\gamma(k-1)+1})^{\frac{\omega}{3}+o(1)}$.

Applying this running time to our setting, this yields an algorithm solving $(k - 1)$ -Multiple k -Dominating Set in the applicable cases in time $\left(n\left(\frac{m}{n}\right)^{k-1}\right)^{\omega/3+o(1)}$. We also complement this with a matching conditional lower bound based on the k -Clique Hypothesis for the $(k - 1)$ -Multiple k -Dominating Set problem.

► **Theorem 5.** *Let G be a graph with n vertices and $m = \Theta(n^{1+\gamma})$ edges for any rational $0 \leq \gamma \leq 1$. For any $\varepsilon > 0$, an algorithm solving $(k - 1)$ -Multiple k -Dominating Set on G in time $\mathcal{O}\left(\left(n\left(\frac{m}{n}\right)^{k-1}\right)^{\omega/3-\varepsilon}\right)$ would refute the k -Clique Hypothesis.*

³ For a definition of the 3-uniform Hyperclique Hypothesis, we refer to Section 2.

⁴ For more details, we refer to Section 2 and Section 3.

Dominating Patterns in Graphs. For graphs G and H , we say a subset $S \subseteq V(G)$ is an H -Dominating Set if S dominates G and induces a subgraph of G that is isomorphic to H . For a fixed constant k , we define the k -Pattern Dominating Set Problem as follows: Given a graph G with n vertices and m edges and a graph H with k vertices, decide if G contains an H -Dominating Set. We can observe that this problem is at most as hard as listing all dominating sets of size k in a graph and hence we can solve it in $mn^{k-2+o(1)}$ for all sufficiently large k . On the other hand, it has been implicitly proved in [29] that detecting the patterns isomorphic to the star graph $K_{1,(k-1)}$ that dominate G takes at least $mn^{k-2-o(1)}$, unless k -OV hypothesis fails, hence settling the fine-grained complexity of k -Pattern Domination in sparse graphs. A more interesting direction is to ask what happens to the complexity of this problem if H is a fixed graph rather than a part of the input. We call this problem H -Dominating Set Problem. In the context of graph theory, such variants of dominating sets have been widely studied, for a variety of natural choices of pattern H that include:

- Dominating Clique [14, 18, 21, 38, 39, 10, 40]
- Dominating Independent Set [15, 16, 17, 41, 45]
- Dominating Path [25, 26, 51]
- Dominating Cycle [23, 24].

Notably, the Dominating Independent Set problem is equivalent to the well-known Maximal Independent Set problem [7, 8, 9, 33]. It turns out that the fine-grained complexity of the H -Dominating Set problem in sparse graphs depends heavily on the choice of H . Obviously, for any fixed H this problem is at most as hard as the k -Pattern Dominating Set Problem and hence can be solved in time $mn^{k-2+o(1)}$. As our first contribution for this problem we show that for no k -vertex graph H and $\varepsilon > 0$, can we solve this problem in the running time $\mathcal{O}\left(\left(\frac{m^{(k+1)/2}}{n}\right)^{1-\varepsilon}\right)$, unless k -OV hypothesis fails.

► **Theorem 6.** *Let H be any graph on $k \geq 3$ vertices. For no $\varepsilon > 0$ is there an algorithm solving H -Dominating Set in time $\mathcal{O}\left(\left(\frac{m^{(k+1)/2}}{n}\right)^{1-\varepsilon}\right)$, unless the k -OV hypothesis fails.*

We then consider the two most studied patterns, namely k -Clique and k -Independent Set. For sufficiently large k , by leveraging the simple fact that there are at most $\mathcal{O}(m^{k/2})$ many cliques in the graph with m edges as well as fast matrix multiplication, we can obtain an algorithm for Dominating k -Clique problem running in time $\left(\frac{m^{(k+1)/2}}{n}\right)^{1+o(1)}$, thus matching the lower bound from Theorem 6.

► **Theorem 7.** *Let $k \geq 5$ be a fixed constant. The Dominating k -Clique problem can be solved on graphs with n vertices and m edges in time*

$$\text{MM}\left(\frac{m}{n} \cdot m^{\frac{1}{2}\lfloor \frac{k-1}{2} \rfloor}, n, m^{\frac{1}{2}\lceil \frac{k-1}{2} \rceil}\right).$$

where $\text{MM}(a, b, c)$ is the time required to multiply an $a \times b$ matrix by a $b \times c$ matrix. If $\omega = 2$, this becomes $\left(\frac{m^{(k+1)/2}}{n}\right)^{1+o(1)}$.

On the other hand, the number of independent sets of size k in sparse graphs is typically much larger and can be as large as $\Theta(n^k)$. Still, perhaps surprisingly, by leveraging some simple structural properties of maximal independent sets, we can obtain an algorithm matching the lower bound of Theorem 6.

► **Theorem 8.** *Let $k \geq 3$ be fixed. The Dominating k -Independent Set problem can be solved on graphs with n vertices and m edges in time $\left(\frac{m^{(k-1+\omega)/2}}{n}\right)^{1+o(1)}$. If $\omega = 2$, this becomes $\left(\frac{m^{(k+1)/2}}{n}\right)^{1+o(1)}$.*

So far we mentioned the full classification of three structurally very different choices of patterns H , that all fall into one of the two extreme regimes of being either as hard as the general k -Pattern Domination problem, or being as easy as any pattern can be. This raises a question if we could provide a fine-grained dichotomy for this class of problems by showing that for each pattern H , the conditionally optimal running time for solving H -Dominating Set problem is either $mn^{k-2\pm o(1)}$, or $\left(\frac{m^{(k+1)/2}}{n}\right)^{1+o(1)}$.

As our last contribution, we answer this question negatively (assuming k -OV hypothesis) by tightly classifying the k -Induced Matching Domination problem that lies in neither of those two regimes, unless k -OV hypothesis fails. More precisely, we show that this problem can be solved in running time $m^{\frac{k}{2}+o(1)}$ for all sufficiently large k , and provide a simple matching conditional lower bound by adapting the reduction from Theorem 6.

2 Preliminaries

Let n be a positive integer. We denote by $[n]$ the set $\{1, \dots, n\}$. If S is an n -element set and $0 \leq k \leq n$ is an integer, then $\binom{S}{k}$ denotes the set of all k -element subsets of S .

Let $\omega < 2.3716$ [53] denote the optimal exponent of multiplying two $n \times n$ matrices and $\text{MM}(a, b, c)$ the time required to multiply two rectangular matrices of dimensions $a \times b$ and $b \times c$. Note that if $\omega = 2$, $\text{MM}(a, b, c) \leq (ab + ac + bc)^{1+o(1)}$. Let $\mathbb{Z}_{\leq d}[X]$ denote the set of all polynomials of degree at most d whose coefficients are integers. For a polynomial $f \in \mathbb{Z}_{\leq d}[X]$, the (*maximum*) *degree* of f is the largest exponent r such that the term X^r has a non-zero coefficient in f . Symmetrically, the (*minimum*) *degree* of f denotes the smallest exponent r such that the term X^r has a non-zero coefficient in f .

For a graph G and a vertex $v \in V(G)$, the *neighbourhood* of v is the set of vertices adjacent to v , denoted $N(v)$. The *closed neighbourhood* of v , denoted $N[v]$ is defined as $N[v] := N(v) \cup \{v\}$. For the subset $S \subseteq V(G)$, we denote $N(S) := \bigcup_{v \in S} N(v)$ (resp. $N[S] := \bigcup_{v \in S} N[v]$). The *degree* of v denotes the size of its neighbourhood ($\deg(v) = |N(v)|$). We further denote by $\deg[v]$ the size of the closed neighbourhood of the vertex v ($\deg[v] = |N[v]|$). For any two vertices $u, v \in V(G)$, we denote by $d_G(u, v)$ the length of the shortest path between u and v in G . The *clique* (resp. *independent set*) in a graph G is a set of pairwise adjacent (resp. nonadjacent) vertices. The *Unbalanced k -Clique* problem, denoted *k -Clique*(n_1, \dots, n_k) is to decide, given a k -partite graph with the i -th part consisting of n_i vertices, whether G has a clique of size k .

2.1 Hardness Assumptions

Consider the k -Orthogonal Vectors problem (k -OV) that is stated as follows: Given sets $A_1, \dots, A_k \subseteq \{0, 1\}^d$, decide whether there exist vectors $a_1 \in A_1, \dots, a_k \in A_k$ such that for all $t \in [d]$, it holds that $\prod_{i=1}^k a_i[t] = 0$. A simple brute force approach solves the k -OV in time $\mathcal{O}(d \cdot \prod_{i \in [k]} |A_i|)$. On the other hand, it is known that for sufficiently large d (e.g., $d = \log^2(|A_1| + \dots + |A_k|)$), any polynomial improvement over this running time would refute SETH (see [50, 52]).

► **Conjecture 9** (*k -OV Hypothesis*). *For no $\varepsilon > 0$ and for no $0 \leq \gamma_1, \dots, \gamma_k \leq 1$ is there an algorithm solving k -OV with $|A_1| = n^{\gamma_1}, \dots, |A_k| = n^{\gamma_k}$, $d = \log^2 n$ in time $\mathcal{O}(n^{(\sum_{i=1}^k \gamma_i) - \varepsilon})$.*

Typically, the k -OV hypothesis is stated for the special case for $\gamma_1 = \dots = \gamma_k = 1$. However, these two hypotheses are known to be equivalent, in a sense that refuting one would also refute the other [13, 29], and for the purposes of this paper, we benefit from using the more general statement.

The k -Clique Detection problem is to decide, given a graph G with n vertices, if G contains a clique of size k , i.e., a set of k pairwise adjacent vertices. For the special case of $k = 3$, a folklore algorithm based on matrix multiplication detects triangles in time $O(n^\omega)$ time. Moreover, for larger k (divisible by 3), one can solve k -Clique Detection in $O(n^{\omega k/3})$ by a split-and-list reduction to the triangle case [44, 22]. Notably, no improvements over these simple algorithms have been made for decades, thus suggesting that they might be optimal and leading to the following hypothesis (see e.g. [1]).

► **Conjecture 10** (*k -Clique Hypothesis*). *For no $\varepsilon > 0$ and $k \geq 3$ is there an algorithm solving k -Clique Detection in time $\mathcal{O}(n^{k\omega/3-\varepsilon})$.*

The h -Uniform k -Hyperclique Detection problem is to decide, given an h -uniform hypergraph G with n vertices, if G contains a hyperclique of size k (i.e. k vertices x_1, \dots, x_k such that each h -tuple x_{i_1}, \dots, x_{i_h} of pairwise distinct $i_1, \dots, i_h \in [k]$ forms an edge in G). When dealing with h -uniform hypergraphs, it turns out that the same matrix multiplication techniques used for k -Clique Detection cannot be used to improve over brute-force [43]. In fact, for $h \geq 3$, no algorithm is known that would detect a k -hyperclique in an n -vertex h -uniform hypergraphs in time $O(n^{k-\varepsilon})$. Moreover any such improvement would cause a breakthrough for other notoriously hard problems as well, most notably Max- h -SAT (see e.g. [43] for a more comprehensive discussion on the hardness of hyperclique detection). This has led to the following hypothesis.

► **Conjecture 11** (*h -Uniform k -Hyperclique Hypothesis*). *For no $\varepsilon > 0, h \geq 3, k \geq h + 1$ is there an algorithm solving h -Uniform k -Hyperclique Detection in time $O(n^{k-\varepsilon})$.*

For the purposes of this paper, we need a seemingly slightly more general assumption. Specifically, we assume that we cannot detect an h -uniform k -hyperclique in a k -partite graph $G = (V_1 \cup \dots \cup V_k, E)$ with arbitrary set sizes $|V_i|$ significantly faster than brute-force, i.e. in time $\mathcal{O}(\left(\prod_{i \in [k]} |V_i|\right)^{1-\varepsilon})$.

► **Conjecture 12** (*Unbalanced h -Uniform k -Hyperclique Hypothesis*). *For no $\varepsilon > 0, h \geq 3, k \geq h + 1$ is there an algorithm solving h -Uniform k -Hyperclique Detection in k -partite graph $G = (V_1 \cup \dots \cup V_k, E)$, with $|V_1| = n^{\gamma_1}, \dots, |V_k| = n^{\gamma_k}$ in time $\mathcal{O}(n^{(\sum_{i=1}^k \gamma_i)-\varepsilon})$.*

However, it turns out that these two assumptions are equivalent in a sense that refuting the Unbalanced h -Uniform k -Hyperclique Hypothesis would refute the h -Uniform k -Hyperclique Hypothesis and vice-versa. The proof is a straightforward self-reduction and is analogous to the proof that the k -OV Hypothesis is equivalent to the Unbalanced k -OV Hypothesis, see [13, 29].

3 r -Multiple k -Dominating Set

In this section, we provide the algorithms for the r -Multiple k -Dominating Set in sparse graphs. In particular, we prove a refined version of Theorem 1 and prove Theorem 3. We also show that the first algorithm cannot be significantly improved without violating some standard fine-grained hypotheses, by proving Theorem 2 and finally show a conditional lower bound for the second algorithm.

3.1 Algorithms

All of our algorithms leverage the following simple lemma.

► **Lemma 13.** *For any fixed $k \geq 2$ and $r \leq k$, any r -Multiple Dominating Set of size k contains at least r vertices v_1, \dots, v_r with $\deg[v_i] \geq \frac{r}{k}$.*

The proof of the lemma can be found in the full version of the paper. We call a vertex v satisfying $\deg[v] \geq \frac{n}{k}$ a *heavy vertex* and we let \mathcal{H} denote the set of all heavy vertices. A simple counting argument shows that there are at most $\mathcal{O}(\frac{m}{n})$ heavy vertices.

We distinguish between two cases based on the dependence of r and k , namely $r \leq k - 2$ and $r = k - 1$ (note that if $r = k$, the problem becomes trivial), and in both cases we are able to show polynomial improvements in the sparse graphs. Let us first consider the case $r \leq k - 2$.

Case $r \leq k - 2$. To construct the desired algorithm, we modify the approach of [22, 29] by employing polynomials to not only determine if a vertex is dominated by a set D , but also count how many vertices from D are in its closed neighbourhood. We obtain the following refined version of Theorem 1.

► **Theorem 14.** *For any fixed $k \geq 3$ and $r \leq k - 2$, we can solve the r -Multiple k -Dominating Set in time*

$$\text{MM} \left(n^{\lceil \frac{k-r}{2} \rceil} \cdot \left(\frac{m}{n}\right)^{\lfloor \frac{r}{2} \rfloor}, n, n^{\lfloor \frac{k-r}{2} \rfloor} \cdot \left(\frac{m}{n}\right)^{\lceil \frac{r}{2} \rceil} \right).$$

If $\omega = 2$, or if k is sufficiently large, this running time becomes $(m/n)^r n^{k-r+o(1)}$.

Proof. Let \mathcal{S} be the set consisting of all subsets of V of size $\lceil \frac{k-r}{2} \rceil + \lfloor \frac{r}{2} \rfloor$ that contain at least $\lfloor \frac{r}{2} \rfloor$ heavy vertices and \mathcal{T} be the set consisting of all subsets of V of size $\lfloor \frac{k-r}{2} \rfloor + \lceil \frac{r}{2} \rceil$ that contain at least $\lceil \frac{r}{2} \rceil$ heavy vertices. By Lemma 13, any potential r -Multiple Dominating Set of size at most k in G can be written as a union of two elements $S \in \mathcal{S}, T \in \mathcal{T}$. Moreover, as argued above, we can bound the size of \mathcal{S} and \mathcal{T} as $|\mathcal{S}| \leq \mathcal{O}(n^{\lceil \frac{k-r}{2} \rceil} \cdot (\frac{m}{n})^{\lfloor \frac{r}{2} \rfloor})$ and $|\mathcal{T}| \leq \mathcal{O}(n^{\lfloor \frac{k-r}{2} \rfloor} \cdot (\frac{m}{n})^{\lceil \frac{r}{2} \rceil})$. We now construct the matrices A and B as follows. Let the rows of A be indexed by \mathcal{S} and columns by V and set the entry $A[S, v]$ to $x^c \in \mathbb{Z}_{\leq k}[X]$ if and only if there are exactly c elements in S that are in the closed neighbourhood of v . Similarly let B have columns indexed by \mathcal{T} and rows by V and set the entry $B[v, T]$ to $x^c \in \mathbb{Z}[X]$ if and only if there are exactly c elements in T that are in the closed neighbourhood of v .

Let $C := A \cdot B$. We observe that if S, T are disjoint, then the coefficient of x^c in $C[S, T]$ counts the number of vertices in V that are dominated by exactly c vertices from $S \cup T$. Hence, it suffices to verify if there exists a pair $S \in \mathcal{S}, T \in \mathcal{T}$ that are disjoint, such that the minimum degree of the polynomial $C[S, T]$ is $\geq r$. Moreover, it is straightforward to see that the degree of any entry in C is bounded by $k = \mathcal{O}(1)$, and hence we can compute C in the desired running time by applying the fastest matrix multiplication algorithm over the ring $\mathbb{Z}_{\leq k}[X]$. The claimed running time follows. ◀

Case $r = k - 1$. By running the same algorithm as above, we can achieve a running time of $\mathcal{O}((\frac{m}{n})^{k-1+o(1)}n + n^2)$ (assuming $\omega = 2$, or sufficiently large k). However, perhaps surprisingly we can beat this running time significantly for larger k . In fact, we proceed to show that for each $k \geq 3$, we can reduce the $(k - 1)$ -Multiple k -Dominating Set problem to an instance of k -Clique $(\frac{m}{n}, \dots, \frac{m}{n}, n)$. To achieve that, we leverage the following simple observation.

► **Observation 15.** *For any fixed $k \geq 2$, let x_1, \dots, x_k be any $(k - 1)$ -Multiple k -Dominating Set. Then for each $i \neq j$, vertices x_i, x_j form a dominating set.*

Given a graph G , we can exploit this observation to preprocess the graph as follows. Recall that \mathcal{H} denotes the set of heavy vertices in our graph and by Lemma 13, any $(k - 1)$ multiple dominating set of size k contains at least $(k - 1)$ heavy vertices. Let V_1, \dots, V_{k-1} be copies

of \mathcal{H} and V_k a copy of $V(G)$. Let $G' = (V', E')$, where $V' = V_1 \cup \dots \cup V_k$ and for any pair $v_i \in V_i, v_j \in V_j$ (for $i \neq j$), add an edge between v_i, v_j if and only if they form a dominating set in G .

► **Lemma 16.** *Let G' be constructed as above. Then vertices v_1, \dots, v_k form a clique in G' if and only if they form a $(k-1)$ -Multiple k -Dominating Set in G .*

Proof. Assume first that some vertices v_1, \dots, v_k form a clique in G' . We will call these vertices *solution vertices*. Take any vertex $w \in V(G)$ and assume that it is dominated by at most $k-2$ solution vertices. In particular, this means w is not dominated by some pair of solution vertices v_i, v_j . However, this means that v_i, v_j is not a dominating set and consequently, there is no edge between v_i and v_j in G' , contradicting that the solution vertices form a clique. The converse follows directly from Observation 15. ◀

By using the approach from [29], we can list all dominating sets of size 2 in time $m^{\omega/2+o(1)}$.

► **Lemma 17** ([29]). *Given a graph G with n vertices and m edges, there exists a randomized algorithm listing all dominating sets of size 2 in time $m^{\omega/2+o(1)}$.*

This gives us all the tools necessary to prove Theorem 3.

► **Theorem 3.** *For any fixed constant $k \geq 2$, let $T_k(m, n)$ denote the time required to solve the k -Clique $(\frac{m}{n}, \dots, \frac{m}{n}, n)$ problem. There is a randomized algorithm solving $(k-1)$ -Multiple k -Dominating Set in time $m^{\frac{\omega}{2}+o(1)} + \mathcal{O}(T_k(m, n))$.*

Proof. Note that it is sufficient to show that we can construct the graph G' as defined above in time $m^{\frac{\omega}{2}+o(1)}$. Given a graph G , let V_i, V_j be two arbitrary parts of G' as described above. Using the algorithm from Lemma 17, we can construct all the edges between the two parts in time at most $m^{\frac{\omega}{2}+o(1)}$ with high probability. We repeat this for each pair $1 \leq i < j \leq k$. ◀

Interestingly, this procedure also gives a polynomial improvement over brute-force in dense graphs.

► **Corollary 18.** *We can solve $(k-1)$ -Multiple k -Dominating Set in time $\mathcal{O}(n^{\omega \frac{k}{3}+1})$.*

Theorem 3 gives us a useful way to think about our problem in terms of the Unbalanced k -Clique problem. However, the question arises how to optimally solve this variation of k -Clique. We partially answer this question by providing infinitely many values of $0 \leq \gamma \leq 1$, such that if $\frac{m}{n} = \mathcal{O}(n^\gamma)$, then for infinitely many values of k , we can solve this problem in time $((\frac{m}{n})^{k-1} n)^{\frac{\omega}{3}+o(1)}$, which is optimal under the k -Clique Hypothesis (see Subsection 3.2 for details). The idea is to apply the standard technique of grouping the vertices that form smaller cliques into three groups W_1, W_2, W_3 of roughly the same size, in such a way that there is a triangle between any three vertices $w_1 \in W_1, w_2 \in W_2, w_3 \in W_3$ if and only if there are vertices $v_1 \in V_1, \dots, v_k \in V_k$ that form a k -clique. In order to be able to achieve this tightly, the values k and γ need to satisfy certain conditions.

► **Proposition 4.** *Let $G = (V_1, \dots, V_k, E)$ be a k -partite graph with vertex part sizes $|V_1| = n$ and $|V_2| = \dots = |V_k| = n^\gamma$ for some $0 \leq \gamma \leq 1$. If*

1. $(k-1 + \frac{1}{\gamma})$ is an integer divisible by 3,
2. $\frac{2}{\gamma} < k-1$,

then we can decide if G has a clique of size k in time $(n^{\gamma(k-1)+1})^{\frac{\omega}{3}+o(1)}$.

Notice that for each positive integer p , by setting $\gamma = \frac{1}{p}$, the first condition is satisfied by every $k \equiv 2p + 1 \pmod{3}$, and the second condition is satisfied by every sufficiently large k (in particular, for each of the infinitely many such choices of γ , both conditions can be satisfied by any of the infinitely many choices of k). For a detailed proof, see the full version of the paper.

3.2 Lower Bound

In this section, we show that the algorithms provided in the previous section are conditionally optimal. To do so, we introduce an intermediate problem, *r-Multiple k-Orthogonal Vectors* defined as follows.

► **Definition 19** (*r-Multiple k-Orthogonal Vectors*). *Given sets $A_1, \dots, A_k \subseteq \{0, 1\}^d$, determine if there exist vectors $a_1 \in A_1, \dots, a_k \in A_k$ such that for each $t \in [d]$ there are pairwise distinct indices $i_1, \dots, i_r \in [k]$ with $a_{i_1}[t] = \dots = a_{i_r}[t] = 0$.*

We note that when $r = 1$, this problem is exactly the *k-Orthogonal Vectors* problem. We can now adapt the reduction from [29] to show that this problem reduces to a sparse instance of *r-Multiple k-Dominating Set*. We note that we are using the *moderate-dimensional* variant of *r-Multiple k-OV* problem (i.e. $d = n^\delta$ for some small $\delta > 0$).

► **Lemma 20.** *For any fixed $k \geq 2$, $1 \leq r \leq k - 1$, let A_1, \dots, A_k be a given instance of *r-Multiple k-Orthogonal Vectors* with $|A_1| = \dots = |A_r| = \mathcal{O}(\frac{m}{n})$ (for any $n \leq m \leq n^2$) and $|A_{r+1}| = \dots = |A_k| = n$. We can construct in linear time an equivalent instance of *r-Multiple k-Dominating Set* G with $\mathcal{O}(n)$ vertices and $\mathcal{O}(m + dn)$ edges.*

Proof. Given an instance A_1, \dots, A_k of *r-Multiple k-Orthogonal Vectors*, let $V(G) = X_1 \cup \dots \cup X_k \cup D \cup R$ where the set X_i corresponds to the set A_i , $D := [d]$ corresponds to the set of dimensions and R is a set containing $(k + 1) \binom{k}{r}$ vertices, representing “redundant” vertices. For each vertex $x_i \in X_i$ add an edge between x_i and $t \in D$ if and only if the corresponding vector a_i satisfies $a_i[t] = 0$. Partition R into $\binom{k}{r}$ many sets R_Q (for each $Q \in \binom{[k]}{r}$) of size $(k + 1)$, and add an edge between any vertex $x_i \in X_i$ and any vertex $y \in R_Q$ if and only if $i \in Q$. It is straightforward to verify now that if G has an *r-multiple k-dominating set* S , it must satisfy $|S \cap X_i| = 1$ for each $i \in [k]$. Finally for each vertex $x_i \in X_i$ for $i \leq r$ and $x_j \in X_j$ for $j \neq i$ add an edge between x_i and x_j .

It is straightforward to verify that the vectors $a_1 \in A_1, \dots, a_k \in A_k$ satisfy the *r-Multiple k-OV* condition if and only if the corresponding vertices $x_1 \in X_1, \dots, x_k \in X_k$ form an *r-multiple dominating set* in G . It remains to show that the constructed graph has $\mathcal{O}(n)$ many vertices and $\mathcal{O}(m + dn)$ many edges. Clearly, G consists of $\mathcal{O}(r \frac{m}{n} + kn + d) = \mathcal{O}(\frac{m}{n} + n) = \mathcal{O}(n)$ many vertices and there are at most $\mathcal{O}(d(r \frac{m}{n} + kn) + r \frac{m}{n} \cdot kn) = \mathcal{O}(dn + m)$ edges. ◀

► **Corollary 21.** *Let $k \geq 2$, $1 \geq r \leq k - 1$ be fixed and $m = \Theta(n^{1+\gamma})$ for some $0 < \gamma \leq 1$. If we can solve *r-Multiple k-Dominating Set* on graphs with n vertices and m edges in time $T(m, n)$, then there exists a $\delta > 0$, such that we can solve any instance A_1, \dots, A_k of *r-Multiple k-Orthogonal Vectors* with $|A_1| = \dots = |A_r| = \mathcal{O}(\frac{m}{n})$, $A_{r+1} = \dots = A_k = n$ and $d = n^\delta$ in time $\mathcal{O}(T(m, n))$.*

Proof. Let $\delta = \gamma/2$ and given an instance A_1, \dots, A_k of *r-Multiple k-Orthogonal Vectors* with $|A_1| = \dots = |A_r| = \mathcal{O}(\frac{m}{n})$, $A_{r+1} = \dots = A_k = n$ and $d = n^\delta$, apply the reduction from the previous lemma to obtain a graph with $\mathcal{O}(n)$ many vertices and $\mathcal{O}(m)$ many edges and run the algorithm solving *r-Multiple k-Dominating Set* in time $T(m, n)$ to this graph to obtain a $\mathcal{O}(T(m, n))$ algorithm for *r-Multiple k-Orthogonal Vectors*. ◀

It now remains to show that r -Multiple k -Orthogonal Vectors problem is conditionally hard. In order to do this, we leverage the fine-grained classification of the first-order properties provided in [12] (for details see the full version of the paper). This allows us to prove the following result.

► **Lemma 22.** *Let X_1, \dots, X_k be an instance of r -Multiple k -OV for $1 \leq r \leq k - 2$. There is no algorithm solving r -Multiple k -OV for $r \leq k - 2$ in time $\mathcal{O}\left(\left(|X_1| \cdots |X_k|\right)^{1-\varepsilon}\right)$ for any $\varepsilon > 0$, unless the $(k - r + 1)$ -Uniform Hyperclique Hypothesis fails. This holds even when restricted to $|X_i| = \Theta(n^{\gamma_i})$ for an arbitrary choice of $\gamma_1, \dots, \gamma_k \in (0, 1]$.*

Finally, by combining this lemma and Corollary 21, we can now conclude that our first algorithm is conditionally optimal (up to subpolynomial factors and resolution of matrix multiplication exponent) under the 3-Uniform Hyperclique Hypothesis.

► **Theorem 2.** *Let $k \geq 3$, $r \leq k - 2$ be fixed constants, and $\varepsilon > 0$. An algorithm solving r -Multiple k -Dominating Set in time $\mathcal{O}\left(\left(m/n\right)^r n^{k-r-\varepsilon}\right)$ would refute the 3-uniform Hyperclique Hypothesis. This holds even when restricting $m = \Theta(n^{1+\gamma})$ for any $0 < \gamma \leq 1$.*

Moreover, we observe that this implies that in dense graphs ($m = \Theta(n^2)$), there is no algorithm solving r -Multiple k -Dominating Set polynomially faster than brute force as long as $r \leq k - 2$, unless 3-Uniform Hyperclique hypothesis fails.

Notably, however, combining r -Multiple k -OV with the tools from [12] fails to provide a tight lower bound for $(k - 1)$ -Multiple k -Dominating Set in sparse graphs (for dense graphs we do get a tight classification, as discussed in the full version of the paper). Nevertheless, by a careful reduction from the Independent Set problem, we can obtain a desired conditional lower bound. We sketch the reduction here. For the full proof see the full version of the paper.

► **Theorem 23.** *Let $0 < \gamma < 1$ be a rational number. Then solving $(k - 1)$ -Multiple k -Dominating Set on graphs with N vertices and $M = N^{1+\gamma}$ edges in time $\mathcal{O}\left(\left(N^{\gamma(k-1)+1}\right)^{\omega/3-\varepsilon}\right)$ for any $\varepsilon > 0$ would refute the k -Clique Hypothesis.*

Proof sketch. Write $\gamma = p/q$ for coprime positive integers p, q and let $k^* := 3(k - 1)p + 3q$. We reduce from k^* -Independent Set Detection. Let $G = (X_1, \dots, X_{k^*}, E)$ be a k^* -partite graph with n vertices in each part. For each $i \in [k - 1]$, let A_i be the set consisting of all independent sets of size $3p$ from $X_{(i-1) \cdot 3p+1}, \dots, X_{i \cdot 3p}$ and A_k be the set consisting of all independent sets of size $3q$ from $X_{k^*-3q+1}, \dots, X_{k^*}$. For each $i \in [k]$, let V_i consist of nodes corresponding to the elements in A_i . Let F be the set corresponding to the edge set E of G . We now construct a graph G' as follows. Let $V(G') = V_1 \cup \dots \cup V_k \cup F \cup R$, where R is a gadget of size $\mathcal{O}(1)$ that ensures that if G' contains any $(k - 1)$ -multiple dominating set of size k , it contains exactly one node from each set V_i . We add the remaining edges as follows. For any pair of nodes $v_i \in V_i, v_j \in V_j$ for $i \neq j$, add an edge between them. Finally, add an edge between a node $f \in F$ and $v_i \in V_i$ if and only if none of the vertices contained in the corresponding independent set $a_i \in A_i$ are among the two endpoints of the edge corresponding to f . By setting $N := n^{3q}$, we can verify that G' contains $\mathcal{O}(N)$ nodes and $\mathcal{O}(N^{1+\gamma})$ edges and that it contains a $(k - 1)$ -multiple dominating set of size k if and only if G has an independent set of size k^* . Finally, if there was an algorithm solving $(k - 1)$ -Multiple k -Dominating Set in time $\mathcal{O}\left(\left(N^{\gamma(k-1)+1}\right)^{\omega/3-\varepsilon}\right)$, by running the reduction above, we could solve the k^* -Independent Set problem in time

$$\mathcal{O}\left(\left(N^{\gamma(k-1)+1}\right)^{\omega/3-\varepsilon}\right) = \mathcal{O}\left(\left(n^{3p(k-1)+3q}\right)^{\omega/3-\varepsilon}\right) = \mathcal{O}\left(n^{k^* \omega/3-\varepsilon'}\right)$$

refuting the k -Clique Hypothesis. ◀

4 Dominating Patterns in Sparse Graphs

In this section we consider the Dominating Pattern problem. In particular, we first provide a simple argument that shows that for every pattern H consisting of k vertices (for $k \geq 3$), we can solve this problem in $mn^{k-2+o(1)}$ running time. On the lower bound side, we observe that the literature implicitly proves existence of a pattern H for which this running time is optimal under the k -OV Hypothesis, thus settling the complexity of the case when the pattern H is a part of the input. We then consider the problem of detecting an H -Dominating Set for a fixed k -vertex graph H . To this end, we show that for any fixed pattern H consisting of k vertices, the existence of an $\mathcal{O}\left(\frac{m^{(k-1)/2+1-\varepsilon}}{n}\right)$ -time algorithm for this problem would refute the k -OV Hypothesis. We then show that this general lower bound is matched by a corresponding algorithm for some patterns H . The fine-grained complexity thus depends heavily on the structure of the graph H itself, and we focus our attention to some of the most important patterns.

► **Proposition 24.** *Let $k \geq 7$. The k -Pattern Domination on graphs with n vertices and m edges can be solved in time $\mathcal{O}(mn^{k-2+o(1)})$ (if $\omega = 2$, we can achieve this running time for all $k \geq 3$).*

For a proof, see the full version of the paper. On the other hand, it has been implicitly proved in [29] that if H is isomorphic to a complete bipartite graph $K_{1,(k-1)}$ (i.e. star graph on k vertices), then detecting an H -Dominating Set in time $\mathcal{O}(mn^{k-2-\varepsilon})$ for any $\varepsilon > 0$ would refute the k -OV Hypothesis, and thus in the general case, the algorithm above is the best we can do, up to subpolynomial factors, unless k -OV Hypothesis fails. We summarise this result in the following.

► **Proposition 25** ([29, Theorem 1.2], reformulated). *Let H be a star graph on k vertices. Then for no $\varepsilon > 0$ is there an algorithm solving the H -Dominating Set problem in time $\mathcal{O}(mn^{k-2-\varepsilon})$, unless the k -OV Hypothesis fails.*

The previous two propositions settle the fine-grained complexity of k -Pattern Domination in sparse graphs, but leave open an interesting research direction. Namely, are there fixed patterns H for which we can beat this running time, and if so, by how much. Towards answering this question, we first provide a conditional lower bound, showing that for no pattern H can we do better than $\left(\frac{m^{1+(k-1)/2}}{n}\right)^{1-o(1)}$ under the k -OV hypothesis.

► **Theorem 6.** *Let H be any graph on $k \geq 3$ vertices. For no $\varepsilon > 0$ is there an algorithm solving H -Dominating Set in time $\mathcal{O}\left(\left(\frac{m^{(k+1)/2}}{n}\right)^{1-\varepsilon}\right)$, unless the k -OV hypothesis fails.*

We adapt the reduction by Fischer, Künnemann and Redzic [29] to force any dominating set of size k to induce the graph H . For a detailed proof see the full version of the paper.

4.1 Dominating k -Clique and k -Independent Set

In this section we consider the two classic graph patterns for which this problem has been well-studied, namely, k -Clique and k -Independent Set. Particularly, we settle the fine-grained complexity of both Dominating k -Clique and Dominating k -Independent Set by providing algorithms that match the conditional lower bound from Theorem 6 even in sparse graphs. Let us focus on the k -Clique case first. In order to obtain a faster algorithm in sparse graphs, we leverage the following observation (for a proof see the full version).

► **Observation 26** (folklore). *A graph with n vertices and m edges has at most $\mathcal{O}(m^{\frac{k}{2}})$ k -cliques.*

► **Theorem 7.** *Let $k \geq 5$ be a fixed constant. The Dominating k -Clique problem can be solved on graphs with n vertices and m edges in time*

$$\text{MM}\left(\frac{m}{n} \cdot m^{\frac{1}{2} \lfloor \frac{k-1}{2} \rfloor}, n, m^{\frac{1}{2} \lceil \frac{k-1}{2} \rceil}\right).$$

where $\text{MM}(a, b, c)$ is the time required to multiply an $a \times b$ matrix by a $b \times c$ matrix. If $\omega = 2$, this becomes $\left(\frac{m^{(k+1)/2}}{n}\right)^{1+o(1)}$.

The idea of the proof is to combine Observation 26 with the observation that each dominating k -clique contains a heavy vertex, in order to restrict the size of our solution space. After restricting the solution space, we follow the similar lines of the matrix multiplication algorithm for k -Dominating Set from [22, 29]. For a detailed proof, see the full version.

The last theorem shows that considering the density of the dominating pattern can be beneficial in obtaining a significant speedup over the standard k -Dominating Set algorithm, by observing that there are fewer such dense patterns (e.g. k -cliques) in sparse graphs. On the other extreme of the density spectrum lie the independent sets. There are typically many k -independent sets in sparse graphs ($\Omega(n^k)$), so we cannot use Observation 26 to obtain a faster algorithm for the Dominating k -Independent Set problem. To nevertheless obtain a fast algorithm, we take advantage of one simple observation. Namely, if we know that some fixed vertex v is contained in some dominating k -independent set, by removing $N[v]$ from G , we can recursively obtain an instance of the Dominating $(k-1)$ -Independent Set problem, since no solution vertices will appear in $N[v]$ and moreover, $N[v]$ is already dominated by v . As a technical note, the crucial reason why this approach fails for instances of the usual k -Dominating Set problem (without the restriction that the solution vertices induce a k -Independent Set) lies in the distinction between monochromatic and bichromatic versions of Dominating Set. In particular, after fixing a solution vertex v of a dominating set, we no longer have to dominate the vertices from $N[v]$, but some of them might still appear in our solution. We thus obtain an instance of Bichromatic $(k-1)$ -Dominating Set (essentially a graph formulation of k -Set Cover), and it is known that this problem is hard already in very sparse instances (see [29]).

► **Lemma 27.** *Let $A_k(G)$ be an algorithm that finds a dominating k -independent set. Given a graph G , any dominating k -independent set containing a fixed vertex v can be found by running $A_{k-1}(G - N[v])$.*

This gives rise to a simple recursive algorithm whose time complexity we can bound as follows. (For a formal proof, we refer to the full version).

► **Lemma 28.** *Let $T_k(n, m)$ denote the running time of an algorithm solving Dominating k -Independent Set problem and \mathcal{H} denote the set of heavy vertices. Then for every $k \geq 3$, the following inequality holds:*

$$T_k(n, m) \leq \sum_{v \in \mathcal{H}} T_{k-1}(n - |N[v]|, m).$$

As the base case of our algorithm, we take the $k = 2$ case, which we can solve in randomized time $m^{\omega/2+o(1)}$.

► **Lemma 29.** *There exists a randomized algorithm solving Dominating 2-Independent Set in time $m^{\omega/2+o(1)}$.*

Proof. By Lemma 17, we can list all dominating sets of size 2 in time $m^{\omega/2+o(1)}$ and for each we can in $\mathcal{O}(1)$ time check if it forms an independent set. ◀

We can now give a full algorithm and its analysis by exploiting the previous lemmas.

► **Theorem 8.** *Let $k \geq 3$ be fixed. The Dominating k -Independent Set problem can be solved on graphs with n vertices and m edges in time $(\frac{m^{(k-1+\omega)/2}}{n})^{1+o(1)}$. If $\omega = 2$, this becomes $(\frac{m^{(k+1)/2}}{n})^{1+o(1)}$.*

Proof. If $k = 2$ we apply Lemma 29 to solve the problem in $m^{\omega/2+o(1)}$. For larger k , for each heavy vertex v , we ask if $G - N[v]$ contains an independent set of size $(k-1)$ that dominates G . If for any choice of v the recursive algorithm returns YES, we return YES and otherwise return NO. We only have to analyse the time complexity. We know by the previous lemma that $T_2(n, m) = m^{\omega/2+o(1)}$. For larger values of k , we have

$$\begin{aligned} T_k(n, m) &\leq \sum_{v \in \mathcal{H}} T_{k-1}(n - |N[v]|, m) \\ &\leq \min(n, \frac{m}{n}) \cdot \max_{v \in \mathcal{H}} T_{k-1}(n - |N[v]|, m) \\ &\leq \frac{m}{n} \cdot \max_{\delta \in [0,1]} T_{k-1}(n^\delta, m) \end{aligned}$$

Now it only remains to bound the value $\max_{\delta \in [0,1]} T_{k-1}(n^\delta, m)$. If $k \leq 3$, this value is bounded by $m^{\omega/2+o(1)}$ and we obtain the claimed running time. So assume that $k \geq 4$ and consider two separate cases, namely when $n^\delta < \sqrt{m}$ and when $n^\delta \geq \sqrt{m}$. In the former case, we can simply list all dominating sets of size $k-1$ in time $n^{k-1+o(1)} < m^{\frac{k-1}{2}+o(1)}$ (assuming $\omega = 2$) using the algorithm from [46]⁵, and this again yields a running time of $(\frac{m^{(k-1+\omega)/2}}{n})^{1+o(1)}$. In the latter case we can proceed inductively, since $\frac{m}{n^\delta} \leq \sqrt{m}$, we have that $T_{k-1}(n^\delta, m) \leq \sqrt{m} \max_{\delta' \in [0,1]} T_{k-2}(n^{\delta'}, m)$ and we can bound $\max_{\delta' \in [0,1]} T_{k-2}(n^{\delta'}, m) \leq m^{(k-2)/2}$ by a simple induction on k , yielding the desired running time. ◀

4.2 Dominating k -Induced Matching

So far we considered three different pattern classes (cliques, independent sets and stars), and in two out of those three cases we can obtain an algorithm that runs in $(\frac{m^{(k+1)/2}}{n})^{1+o(1)}$ (if $\omega = 2$), which is the best we can do for any pattern assuming k -OV Hypothesis, and in the remaining case we can show an $mn^{k-2-o(1)}$ conditional lower bound, which makes this pattern as hard as any pattern can be. This suggests that there might be a dichotomy of all k -vertex graphs into two classes:

1. Easy Patterns (those for which there exists an algorithm solving H -Dominating Set in $(\frac{m^{(k+1)/2}}{n})^{1+o(1)}$)
2. Hard Patterns (those for which we can show an $mn^{k-2-o(1)}$ lower bound under k -OV Hypothesis).

In this section we show that such a dichotomy is unlikely. More precisely, we find a pattern which is in neither of those two categories (unless k -OV Hypothesis fails). Let the k -induced matching be the graph consisting of $k/2$ independent edges. In this section we prove that we can solve the Dominating k -Induced Matching problem in $m^{k/2+o(1)}$ running time (if $\omega = 2$) and provide a matching conditional lower bound under the k -OV Hypothesis. Here we only state our results; the full proofs can be found in the full version.

⁵ If $k \geq 8$, we can obtain this running time, even with the current value of ω .

► **Theorem 30.** *Given a graph G with n vertices and m edges, we can solve Dominating k -Induced Matching in time*

$$\text{MM}(m^{\lceil \frac{k}{2} \rceil}, n, m^{\lfloor \frac{k}{2} \rfloor}).$$

If $\omega = 2$, this running time becomes $m^{\frac{k}{2} + o(1)}$ for every even $k \geq 4$.

Finally, we show that this running time cannot be significantly improved, unless k -OV Hypothesis fails. To achieve this, we apply a simple adaptation of the reduction from Theorem 6.

► **Theorem 31.** *For no even $k \geq 4$ and $\varepsilon > 0$ is there an algorithm solving Dominating k -Induced Matching in time $\mathcal{O}(m^{\frac{k}{2} - \varepsilon})$, unless the k -OV hypothesis fails.*

References

- 1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the current clique algorithms are optimal, so is Valiant’s parser. *SIAM Journal on Computing*, 47(6):2527–2555, 2018. doi:10.1137/16M1061771.
- 2 José D. Alvarado, Simone Dantas, Elena Mohr, and Dieter Rautenbach. On the maximum number of minimum dominating sets in forests. *Discret. Math.*, 342(4):934–942, 2019. doi:10.1016/J.DISC.2018.11.025.
- 3 Toru Araki. The k -tuple twin domination in de Bruijn and Kautz digraphs. *Discret. Math.*, 308(24):6406–6413, 2008. doi:10.1016/J.DISC.2007.12.020.
- 4 Toru Araki. Connected k -tuple twin domination in de Bruijn and Kautz digraphs. *Discret. Math.*, 309(21):6229–6234, 2009. doi:10.1016/J.DISC.2009.05.031.
- 5 G Argiroffo, V Leoni, and P Torres. Complexity of k -tuple total and total $\{k\}$ -dominations for some subclasses of bipartite graphs. *Information Processing Letters*, 138:75–80, 2018. doi:10.1016/J.IPL.2018.06.007.
- 6 Gabriela Argiroffo, Valeria Leoni, and Pablo Torres. On the complexity of $\{k\}$ -domination and k -tuple domination in graphs. *Information Processing Letters*, 115(6-8):556–561, 2015. doi:10.1016/J.IPL.2015.01.007.
- 7 Sepehr Assadi, Gillat Kol, and Zhijun Zhang. Rounds vs communication tradeoffs for maximal independent sets. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 – November 3, 2022*, pages 1193–1204. IEEE, 2022. doi:10.1109/FOCS54457.2022.00115.
- 8 Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear update time. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 815–826. ACM, 2018. doi:10.1145/3188745.3188922.
- 9 Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear in n update time. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1919–1936. SIAM, 2019. doi:10.1137/1.9781611975482.116.
- 10 Alkida Balliu, Sebastian Brandt, Fabian Kuhn, and Dennis Olivetti. Distributed maximal matching and maximal independent set on hypergraphs. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 2632–2676. SIAM, 2023. doi:10.1137/1.9781611977554.CH100.

- 11 Silvia M. Bianchi, Graciela L. Nasini, Paola B. Tolomei, and Luis Miguel Torres. On dominating set polyhedra of circular interval graphs. *Discret. Math.*, 344(4):112283, 2021. doi:10.1016/J.DISC.2020.112283.
- 12 Karl Bringmann, Nick Fischer, and Marvin Künnemann. A fine-grained analogue of Schaefer’s Theorem in P: Dichotomy of $\exists^k\forall$ -quantified First-Order Graph Properties. In *34th computational complexity conference (CCC 2019)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.
- 13 Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 79–97, 2015. doi:10.1109/FOCS.2015.15.
- 14 Victor Chepoi. A note on r -dominating cliques. *Discrete mathematics*, 183(1-3):47–60, 1998. doi:10.1016/S0012-365X(97)00076-9.
- 15 Eun-Kyung Cho, Ilkyoo Choi, Hyemin Kwon, and Boram Park. A tight bound for independent domination of cubic graphs without 4-cycles. *Journal of Graph Theory*, 104(2):372–386, 2023. doi:10.1002/JGT.22968.
- 16 Eun-Kyung Cho, Ilkyoo Choi, and Boram Park. On independent domination of regular graphs. *Journal of Graph Theory*, 103(1):159–170, 2023. doi:10.1002/JGT.22912.
- 17 Eun-Kyung Cho, Jinha Kim, Minki Kim, and Sang-il Oum. Independent domination of graphs with bounded maximum degree. *Journal of Combinatorial Theory, Series B*, 158:341–352, 2023. doi:10.1016/J.JCTB.2022.10.004.
- 18 Margaret B Cozzens and Laura L Kelleher. Dominating cliques in graphs. *Discrete Mathematics*, 86(1-3):101–116, 1990. doi:10.1016/0012-365X(90)90353-J.
- 19 Jonathan Cutler and A. J. Radcliffe. Counting dominating sets and related structures in graphs. *Discret. Math.*, 339(5):1593–1599, 2016. doi:10.1016/J.DISC.2015.12.011.
- 20 Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: basic results. *SIAM J. Comput.*, 24(4):873–921, 1995. doi:10.1137/S0097539792228228.
- 21 Feodor F Dragan and Andreas Brandstädt. Dominating cliques in graphs with hypertree structure. In *STACS 94: 11th Annual Symposium on Theoretical Aspects of Computer Science Caen, France, February 24–26, 1994 Proceedings 11*, pages 735–746. Springer, 1994. doi:10.1007/3-540-57785-8_186.
- 22 Friedrich Eisenbrand and Fabrizio Grandoni. On the complexity of fixed parameter clique and dominating set. *Theoretical Computer Science*, 326(1-3):57–67, 2004. doi:10.1016/J.TCS.2004.05.009.
- 23 Yibin Fang and Liming Xiong. Circumference of a graph and its distance dominating longest cycles. *Discrete Mathematics*, 344(2):112196, 2021. doi:10.1016/j.disc.2020.112196.
- 24 Yibin Fang and Liming Xiong. On the dominating (induced) cycles of iterated line graphs. *Discrete Applied Mathematics*, 325:43–51, 2023. doi:10.1016/j.dam.2022.09.025.
- 25 Jill R. Faudree, Ralph J. Faudree, Ronald J. Gould, Paul Horn, and Michael S. Jacobson. Degree sum and vertex dominating paths. *J. Graph Theory*, 89(3):250–265, 2018. doi:10.1002/JGT.22249.
- 26 Ralph J. Faudree, Ronald J. Gould, Michael S. Jacobson, and Douglas B. West. Minimum degree and dominating paths. *J. Graph Theory*, 84(2):202–213, 2017. doi:10.1002/JGT.22021.
- 27 John Frederick Fink and Michael S Jacobson. n -domination in graphs. In *Graph theory with applications to algorithms and computer science*, pages 283–300, 1985.
- 28 John Frederick Fink and Michael S Jacobson. On n -domination, n -dependence and forbidden subgraphs. In *Graph theory with applications to algorithms and computer science*, pages 301–311, 1985.
- 29 Nick Fischer, Marvin Künnemann, and Mirza Redzic. The effect of sparsity on k -dominating set and related first-order graph properties. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4704–4727. SIAM, 2024. doi:10.1137/1.9781611977912.168.

- 30 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Linear kernels for (connected) dominating set on H -minor-free graphs. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 82–93. SIAM, 2012. doi:10.1137/1.9781611973099.7.
- 31 Fedor V. Fomin and Dimitrios M. Thilikos. Dominating sets in planar graphs: branch-width and exponential speed-up. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA*, pages 168–177. ACM/SIAM, 2003. URL: <http://dl.acm.org/citation.cfm?id=644108.644138>.
- 32 Andrei Gagarin and Vadim E. Zverovich. A generalised upper bound for the k -tuple domination number. *Discret. Math.*, 308(5-6):880–885, 2008. doi:10.1016/J.DISC.2007.07.033.
- 33 Mohsen Ghaffari. Local computation of maximal independent set. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 – November 3, 2022*, pages 438–449. IEEE, 2022. doi:10.1109/FOCS54457.2022.00049.
- 34 Frank Harary and Teresa W. Haynes. Nordhaus-Gaddum inequalities for domination in graphs. *Discrete Mathematics*, 155(1):99–105, 1996. Combinatorics. doi:10.1016/0012-365X(94)00373-Q.
- 35 Frank Harary and Teresa W. Haynes. Double domination in graphs. *Ars Comb.*, 55, 2000.
- 36 Michael A. Henning and Adel P. Kazemi. k -tuple total domination in graphs. *Discret. Appl. Math.*, 158(9):1006–1011, 2010. doi:10.1016/J.DAM.2010.01.009.
- 37 Karthik C. S., Bundit Laekhanukit, and Pasin Manurangsi. On the parameterized complexity of approximating dominating set. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 1283–1296. ACM, 2018. doi:10.1145/3188745.3188896.
- 38 Ekkehard Köhler. Connected domination and dominating clique in trapezoid graphs. *Discret. Appl. Math.*, 99(1-3):91–110, 2000. doi:10.1016/S0166-218X(99)00127-4.
- 39 Dieter Kratsch. Finding dominating cliques efficiently, in strongly chordal graphs and undirected path graphs. *Discrete mathematics*, 86(1-3):225–238, 1990. doi:10.1016/0012-365X(90)90363-M.
- 40 Dieter Kratsch, Peter Damaschke, and Anna Lubiw. Dominating cliques in chordal graphs. *Discrete Mathematics*, 128(1-3):269–275, 1994. doi:10.1016/0012-365X(94)90118-X.
- 41 Kirsti Kuenzel and Douglas F Rall. On independent domination in direct products. *Graphs and Combinatorics*, 39(1):7, 2023. doi:10.1007/S00373-022-02600-0.
- 42 Peng Li, Aifa Wang, and Jianhui Shang. A simple optimal algorithm for k -tuple dominating problem in interval graphs. *J. Comb. Optim.*, 45(1):14, 2023. doi:10.1007/S10878-022-00932-4.
- 43 Andrea Lincoln, Virginia Vassilevska Williams, and Ryan Williams. Tight hardness for shortest cycles and paths in sparse graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1236–1252. SIAM, 2018. doi:10.1137/1.9781611975031.80.
- 44 Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985.
- 45 Shiwei Pan, Yiming Ma, Yiyuan Wang, Zhiguo Zhou, Jinchao Ji, Minghao Yin, and Shuli Hu. An improved master-apprentice evolutionary algorithm for minimum independent dominating set problem. *Frontiers of Computer Science*, 17(4):174326, 2023. doi:10.1007/S11704-022-2023-7.
- 46 Mihai Pătrașcu and Ryan Williams. On the possibility of faster SAT algorithms. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1065–1075. SIAM, 2010. doi:10.1137/1.9781611973075.86.

- 47 Shay Solomon and Amitai Uzrad. Dynamic $((1+\epsilon) \ln n)$ -approximation algorithms for minimum set cover and dominating set. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 1187–1200. ACM, 2023. doi:10.1145/3564246.3585211.
- 48 Akira Suzuki, Amer E. Mouawad, and Naomi Nishimura. Reconfiguration of dominating sets. *J. Comb. Optim.*, 32(4):1182–1195, 2016. doi:10.1007/S10878-015-9947-X.
- 49 Dmitrii S. Taletskii. Trees with extremal numbers of k -dominating sets. *Discret. Math.*, 345(1):112656, 2022. doi:10.1016/J.DISC.2021.112656.
- 50 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proc. of the International Congress of Mathematicians, ICM'18*, pages 3447–3487, 2018.
- 51 Henk Jan Veldman. Existence of dominating cycles and paths. *Discret. Math.*, 43(2-3):281–296, 1983. doi:10.1016/0012-365X(83)90165-6.
- 52 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. doi:10.1016/J.TCS.2005.09.023.
- 53 Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New bounds for matrix multiplication: from alpha to omega. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 3792–3835. SIAM, 2024. doi:10.1137/1.9781611977912.134.


Subexponential Algorithms for Clique Cover on Unit Disk and Unit Ball Graphs

Tomohiro Koana  

Utrecht University, The Netherlands

Nidhi Purohit  

National University of Singapore, Singapore

Kirill Simonov  

Hasso Plattner Institute, University of Potsdam, Germany

Abstract

In CLIQUE COVER, given a graph G and an integer k , the task is to partition the vertices of G into k cliques. CLIQUE COVER on unit ball graphs has a natural interpretation as a clustering problem, where the objective function is the maximum diameter of a cluster.

Many classical NP-hard problems are known to admit $2^{O(n^{1-1/d})}$ -time algorithms on unit ball graphs in \mathbb{R}^d [de Berg et al., SIAM J. Comp 2018]. A notable exception is the MAXIMUM CLIQUE problem, which admits a polynomial-time algorithm on unit disk graphs and a subexponential algorithm on unit ball graphs in \mathbb{R}^3 , but no subexponential algorithm on unit ball graphs in dimensions 4 or larger, assuming the ETH [Bonamy et al., JACM 2021].

In this work, we show that CLIQUE COVER also suffers from a “curse of dimensionality”, albeit in a significantly different way compared to MAXIMUM CLIQUE. We present a $2^{O(\sqrt{n})}$ -time algorithm for unit disk graphs and argue that it is tight under the ETH. On the other hand, we show that CLIQUE COVER does not admit a $2^{o(n)}$ -time algorithm on unit ball graphs in dimension 5, unless the ETH fails.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Clique cover, diameter clustering, subexponential algorithms, unit disk graphs

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.10

Related Version *Preprint*: <https://arxiv.org/abs/2410.03609>

Funding *Tomohiro Koana*: Supported by the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (project CRACKNP under grant agreement No. 853234).

Kirill Simonov: Supported by DFG Research Group ADYN via grant DFG 411362735.

1 Introduction

Clustering is a general method of partitioning data entries, normally represented by points in the Euclidean space, into clusters with the goal of minimizing a certain similarity function for the points in the same cluster. Many popular similarity objectives such as k -means and k -center are center-based, i.e., the objective function of the cluster is defined in terms of distance to the additionally selected center of the cluster. On the other hand, arguably the most natural similarity measure that is defined solely in terms of distances between the given datapoints, is the maximum diameter of a cluster. That is, the objective function of the clustering is the maximum distance between any pair of points in the same cluster. Formally,



© Tomohiro Koana, Nidhi Purohit, and Kirill Simonov;
licensed under Creative Commons License CC-BY 4.0

19th International Symposium on Parameterized and Exact Computation (IPEC 2024).

Editors: Édouard Bonnet and Paweł Rzażewski; Article No. 10; pp. 10:1–10:17



Leibniz International Proceedings in Informatics

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

we consider the following k -DIAMETER problem: Given a set of points P in the Euclidean space \mathbb{R}^d , and parameters k, D , is there a partitioning of P into disjoint C_1, \dots, C_k , such that for each $j \in [k]$, and each $x, y \in C_j$, $\|x - y\| \leq D$?¹

The k -DIAMETER problem admits a natural geometric interpretation. Consider a set of disks with centers in P and of the same radius $D/2$. The problem asks to partition the disks into k sets so that disks in each set pairwise intersect. Given a graph G and an integer k , let CLIQUE COVER be the problem of partitioning the vertex set of G into k vertex-disjoint cliques. k -DIAMETER in \mathbb{R}^d is thus equivalent to CLIQUE COVER on unit ball graphs in \mathbb{R}^d . Note that CLIQUE COVER is equivalent to k -COLORING on general graphs by taking the complement of the graph; however, unit ball graphs are not closed under complements, therefore CLIQUE COVER on unit ball graphs does not necessarily have the same complexity as k -COLORING on unit ball graphs.

The main question we ask in this work is the following: Does k -DIAMETER in \mathbb{R}^d , or equivalently CLIQUE COVER on d -dimensional unit ball graphs, admit subexponential-time algorithms? Given that CLIQUE COVER is a natural graph problem akin to MAXIMUM CLIQUE and k -COLORING, our question fits into the recent line of advances for algorithms on geometric intersection graphs.

In a seminal work, de Berg et al. [6] gave a framework for $2^{O(n^{1-1/d})}$ -time algorithms on, in particular, d -dimensional unit ball graphs, which covers problems such as MAXIMUM INDEPENDENT SET, DOMINATING SET, and STEINER TREE. At the heart of the framework lies a special kind of tree decomposition, that essentially guarantees that each bag is covered by $O(n^{1-1/d}/\log n)$ cliques. The target problem is then solved via dynamic programming over the decomposition, given that the interaction of the solution with the cliques in the bag could be succinctly represented. For example, in the MAXIMUM INDEPENDENT SET problem the solution can have at most one element per clique, and storing the intersection between the solution and the bag is therefore sufficient for the running time above.

However, CLIQUE COVER stands aside from the problems covered by the framework of de Berg et al., as the interaction between the smallest clique cover and the given clique cover of the bag does not immediately seem to admit a succinct representation. Moreover, one can easily observe that finding the smallest clique cover is still NP-hard even if a clique cover of the graph of constant size is given. Indeed, it is famously NP-hard to determine whether a 4-colorable graph admits a 3-coloring [12, 8], and colorings turn into clique covers under taking the complement of the graph.

Previously in the literature, another problem shown to not exhibit such a “gradually subexponential” behavior was the MAXIMUM CLIQUE problem. Already since the 1990s, a polynomial-time algorithm for MAXIMUM CLIQUE on unit disk graphs was known [5]. Recently, Bonamy et al. [2] have shown that MAXIMUM CLIQUE only admits a subexponential-time algorithm on 3-dimensional unit ball graphs, while no $2^{o(n)}$ -time algorithm is possible in dimension 4, assuming the ETH.

Our results. As the first step, we show a subexponential algorithm for CLIQUE COVER on unit disk graphs. Our starting point is the weighted treewidth approach of de Berg et al. [6]; however, as per the discussion above, on its own this characterization does not seem to be sufficient. Intuitively, the geometric structure of unit disk graphs has to play a role not

¹ Since one can binary search over the value of D , and there are at most $|P|^2$ different distances between the pairs of points, this decision version of the problem is equivalent to the optimization version, up to logarithmic factors in the running time.

only in the decomposition itself, but also in representing the solution with respect to the decomposition. In order to accommodate this, we build upon the classical lemma due to Capoteas, Rote and Woeginger [4], that was rediscovered several times in the literature [7, 13]. Simply put, there always exists an optimal clique cover where all cliques are well-separated, i.e., the convex hulls of the respective disk centers do not intersect. As only constantly many cliques may lie in direct vicinity of another clique in an optimal solution, we can show that there are at most polynomially many possible configurations for each clique in the optimal solution. This characterization, coupled with the dynamic programming approach, results in the following theorem.

► **Theorem 1.** *CLIQUE COVER can be solved in time $2^{O(\sqrt{n})}$ on n -vertex unit disk graphs, when a geometric representation of the graph is given in the input, with bit-length of the vectors bounded by $\text{poly}(n)$.*

Note that recognizing unit disk graphs is, in general, NP-hard [3] and even $\exists\mathbb{R}$ -complete [11], which means that one cannot expect to be able to compute a geometric representation of a given unit disk graph efficiently.

Using the lower bound machinery of de Berg et al. [6], we also observe that the running time above is tight. Moreover, the lower bound holds for higher dimensions as well.

► **Theorem 2.** *Assuming the ETH, CLIQUE COVER on n -vertex unit ball graphs in \mathbb{R}^d does not admit a $2^{o(n^{1-1/d})}$ -time algorithm, for any $d > 1$, even if the geometric representation of polynomial bit-length is given in the input.*

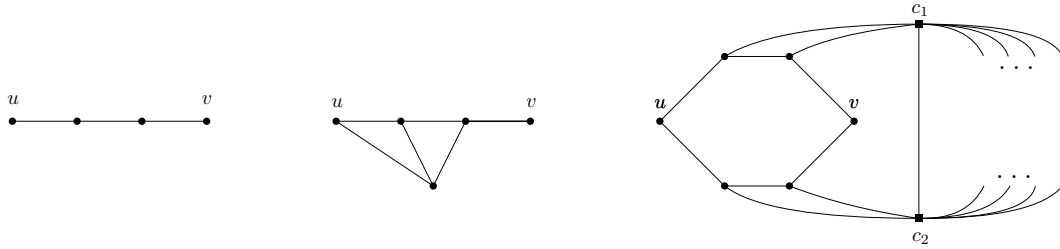
The next natural question is whether the algorithmic result of Theorem 1 could also be extended to higher dimensions. Unfortunately, the separation property that plays the key role in Theorem 1 only holds in the two-dimensional case: the original work of Capoteas, Rote and Woeginger already observes that the analogous statement in three dimensions admits a counterexample [4]. This, however does not exclude other potential ways for a succinct representation of the solution, or another completely unrelated approach. We show that the separation property is indeed crucial, that is, CLIQUE COVER does not admit subexponential algorithms on unit ball graphs in constant dimension.

► **Theorem 3.** *Assuming the ETH, CLIQUE COVER on n -vertex unit ball graphs in \mathbb{R}^5 does not admit a $2^{o(n)}$ -time algorithm, even if the geometric representation of polynomial bit-length is given in the input.*

To put Theorem 3 into context, recall the result of Bonamy et al. [2], showing that MAXIMUM CLIQUE does not admit a subexponential algorithm on unit ball graphs in \mathbb{R}^4 . Their approach is to first argue that MAXIMUM INDEPENDENT SET is as hard on 2-subdivisions (graphs obtained by replacing each edge with a path of length 3) as it is on general graphs, which holds simply because a maximum independent set of a graph can be extracted from a maximum independent set of its 2-subdivision. Then their key structural observation is that a complement of *any* 2-subdivision admits a unit ball representation in \mathbb{R}^4 , therefore showing hardness of MAXIMUM CLIQUE on unit ball graphs in \mathbb{R}^4 . Note that MAXIMUM INDEPENDENT SET turns into MAXIMUM CLIQUE by taking the complement.

Since we target the CLIQUE COVER problem on unit ball graphs, a natural idea is to conduct the reduction in a similar spirit, but starting from k -COLORING. However, the obstacle is that 2-subdivisions do not in general preserve the existence of a k -coloring – only for $k = 2$, which is not suitable for a hardness reduction. Therefore, instead of replacing each edge by its 2-subdivision, we need to use a more complicated edge gadget, and the

4-dimensional representation of Bonamy et al. is no longer applicable. The straightforward triangle-like edge gadget that preserves 3-colorings could be used in place of the 2-subdivision, see Figure 1 for an illustration. However, it is not clear whether the resulting graph would admit a sufficiently low-dimensional representation, namely below dimension 7. Instead, the gadget that we use is based on two parallel 2-subdivisions, plus special vertices that impose a list-coloring-like condition on the internal vertices of the subdivisions; this choice of the gadget allows us to decrease the dimension to 5 (see Figure 1 for the illustration of the gadget).



■ **Figure 1** Edge gadgets encoding the edge between vertices u, v : left, 2-subdivision of the edge, suitable for maximum independent sets; center, triangle-like gadget suitable for 3-colorings; right, improved gadget preserving 3-colorings – here vertices c_1 and c_2 are connected in the same way to all edge gadgets.

2 Preliminaries

Sets, vectors and coordinates

For an integer n , we use $[n]$ to denote the set $\{1, 2, \dots, n\}$. We use the tuple notation for points in \mathbb{R}^d , i.e., a point is defined by the tuple (a_1, a_2, \dots, a_d) , where $a_i \in \mathbb{R}$ is the respective coordinate for each $i \in [d]$. The variables x_1, x_2, \dots, x_d are used to denote the respective axes. We denote the origin by $O = (0, 0, \dots, 0)$, and by $Ox_i x_j$, $i, j \in [d]$ we denote the plane spanned on the respective axes; the same notation is used for higher-dimensional subspaces too. For two points $A, B \in \mathbb{R}^d$, \overrightarrow{AB} denotes the vector pointing from A to B , its coordinates are expressed as $B - A$. We use $\|\cdot\|$ to denote the standard Euclidean norm in \mathbb{R}^2 , therefore, $\|B - A\|$ is the Euclidean distance between the points A and B , and also the length of the vector \overrightarrow{AB} .

Unit ball graphs

Let $P = \{p_1, \dots, p_n\}$ be a set of points in \mathbb{R}^d and B be a set of balls b_i of radius 1, centered at p_i . A unit ball graph on P is a graph over the vertex set P , in which two vertices p_i and p_j are adjacent if and only if the balls b_i and b_j intersect.

Exponential-time hypothesis

The exponential-time hypothesis (ETH), due to Impagliazzo, Paturi and Zane [9, 10], implies that there is no algorithm that solves 3-SAT in $2^{o(n)}$ time, where n is the number of variables in the formula. Since by the Sparsification Lemma [10] this holds even for linearly-many clauses in the formula, ETH also excludes $2^{o(n+m)}$ -time algorithms for 3-SAT, where m is the number of clauses. By the standard linear-size reduction from 3-SAT to 3-COLORING, ETH implies that 3-COLORING does not admit a $2^{o(n+m)}$ -time algorithm, where n is the number of vertices and m is the number of edges in the graph.

Tree decomposition

For a graph $G = (V, E)$, a *tree decomposition* is a pair (T, σ) , where $T = (V_T, E_T)$ is a tree and $\sigma: V_T \rightarrow 2^V$ such that

- for each $uv \in E$, there exists $t \in V_T$ with $u, v \in \sigma(t)$, and
- for each $v \in V$, the set of nodes $t \in V_T$ with $v \in \sigma(t)$ forms a connected subtree in T .

The *width* of (T, σ) is $\max_{x \in V(T)} (|\sigma(x)| - 1)$. The *tree-width* of G is the minimum width of all tree decompositions of G .

A *nice* tree decomposition is a tree decomposition more amenable to the design of dynamic programming algorithms. Formally, a tree decomposition $(T = (V_T, E_T), \sigma)$ rooted at $r \in V_T$ is called *nice* if $\sigma(r) = \emptyset$ and each node $t \in V_T$ is one of the following types:

Leaf node. t is leaf in T and $\sigma(t) = \emptyset$.

Introduce node. t has exactly one child t' , and $\sigma(t) = \sigma(t') \cup \{v\}$ for a vertex v in G .

Forget node. t has exactly one child t' , and $\sigma(t) = \sigma(t') \setminus \{v\}$ for a vertex v in G .

Join node. t has exactly two children t', t'' , and $\sigma(t) = \sigma(t') = \sigma(t'')$.

It is known that given a tree decomposition, a nice tree decomposition of the same width can be computed in polynomial time [1].

3 Subexponential algorithm for unit disks

In this section, we design a subexponential-time algorithm for CLIQUE COVER on unit disk graphs.

► **Theorem 1.** *CLIQUE COVER can be solved in time $2^{O(\sqrt{n})}$ on n -vertex unit disk graphs, when a geometric representation of the graph is given in the input, with bit-length of the vectors bounded by $\text{poly}(n)$.*

To design a subexponential-time algorithm, let us introduce two known techniques. We start with the “separation theorem” of Capoteas, Rote and Woeginger [4]. Recall that we aim to partition the vertex set of a given unit disk graph into a collection of k cliques. Each clique is defined by the convex hull of the centers of disks in the clique. In principle, these convex hulls may arbitrarily intersect each other. The following states that we may assume that they are disjoint in an optimal solution.

► **Theorem 4** (Capoteas, Rote and Woeginger [4]). *For CLIQUE COVER on unit disk graphs, there exists an optimal solution (C_1, \dots, C_ℓ) such that the convex hulls of the centers in C_i are pairwise disjoint.*

This was first proven by Capoteas, Rote and Woeginger [4] but also by Dumitrescu and Pach [7] and Pirwani and Salavatipour [13] later. Theorem 4 relies crucially on the fact that for two intersecting convex polygons P_1, P_2 of diameter at most d , there exists two disjoint convex polygons P'_1, P'_2 of diameter at most d such that the vertices of P_1 and P_2 are contained in $P'_1 \cup P'_2$. In view of Theorem 4, we will show that there are polynomially many “relevant” cliques in Lemma 6. To prove this, we will also use the following simple fact.

► **Lemma 5** (Dumitrescu and Pach, Lemma 2 [7]). *Let (G, ℓ) be an instance of CLIQUE COVER on unit disk graphs, and (C_1, \dots, C_ℓ) be an optimal solution satisfying the condition of Theorem 4. For a set S of vertices contained in a square of constant side length, there are $O(1)$ cliques C_i that intersect S .*

See Dumitrescu and Pach [7] for a concrete bound in the above lemma. Now we prove a polynomial bound on the number of relevant cliques.

► **Lemma 6.** *Let (G, ℓ) be an instance of CLIQUE COVER on unit disk graphs. Given $S \subseteq V(G)$, we can find in polynomial time a collection \mathcal{R} of cliques in G such that $|\mathcal{R}| \in |S|^{O(1)}$, and for each optimal solution (C_1, \dots, C_ℓ) satisfying the condition of Theorem 4, $S \cap C_i \in \mathcal{R}$ for all $i \in [\ell]$.*

Proof. Let $C = S \cap C_i$ be a clique with $C \neq \emptyset$. We will say that a clique C_j is *close* to C if their closest vertices have distance at most two and *far* from C otherwise.

We first show how to separate C from far cliques, i.e., we find a collection of closed regions P such that C lies within P and any far clique lies outside P . Suppose that $u, v \in C$ are two vertices with the largest distance $r \leq 2$ in C . Then, C is contained in the intersection of two disks of radius r centered at u and v , and every vertex of every far clique from C is outside of these disks. For each $u, v \in S$, let $P_{u,v}$ be the intersection of such two disks, and let $R_{u,v}$ be the vertices of S that lie in $P_{u,v}$. Let \mathcal{R}' be the collection of vertex sets containing $R_{u,v}$ for each $u, v \in S$. We then have $|\mathcal{R}'| \in O(|S|^2)$, and for every $C = S \cap C_i$ there exists $R \in \mathcal{R}'$ that does not intersect any clique far from C .

Next, we discuss how to separate C from close cliques. By the above characterization, C is contained in a 2×4 -rectangle (not necessarily axis-aligned). For each close clique C' of C , there exists a vertex t in C' with distance at most 2 to a vertex in C , and every vertex in C' has distance at most 2 to t , so every vertex of C' is at most at distance 4 from some vertex of C . Therefore by extending the 2×4 rectangle containing C by 4 in every direction, we obtain a 10×12 rectangle that contains every close clique of C . Thus, by Lemma 5, there are $O(1)$ close cliques C_j with $j \in [\ell]$. For a clique C_j , $j \in [\ell]$, since the convex hulls of C and C_j do not overlap by Theorem 4, there is a line that separates C and C_j , this line also separates the convex hulls of C and $C_j \cap S$. Moving this line, we find two vertices on the boundary of the convex hull of C or two vertices on the boundary of the convex hull of $S \cap C_j$, such that the line through them separates C and $S \cap C_j$ in the plane. Let \mathcal{P}'' be the collection of regions obtained as the intersection of constantly² many open or closed semi-planes whose boundaries go through two points of S . Let \mathcal{R}'' be the collection of vertex sets such that for each region in \mathcal{P}'' , there is a vertex set in \mathcal{R}'' containing exactly the vertices of S lying in this region.

Finally, let \mathcal{R} be the collection of intersections of R' and R'' for $R' \in \mathcal{R}'$ and $R'' \in \mathcal{R}''$. Clearly, $|\mathcal{R}| \in |S|^{O(1)}$. By the above, we have that for $C = S \cap C_i$, there exists $R' \in \mathcal{R}'$ that is disjoint from $S \cap C_j$ for every clique C_j that is far from C , and there exists $R'' \in \mathcal{R}''$ that is disjoint from $S \cap C_h$ for every clique C_h close to C ; on the other hand, R' and R'' contain C . Therefore, $R' \cap R''$ is disjoint from $S \cap C_j$ for every $j \neq i$, and contains C . Since $V(G) = C_1 \cup \dots \cup C_\ell$ and $R', R'' \subseteq S \subseteq V(G)$, $R' \cap R''$ contains no vertices outside of C , and $C = R' \cap R'' \in \mathcal{R}$. Moreover, every $R \in \mathcal{R}$ is a clique since every $R' \in \mathcal{R}'$ is a clique. This completes the proof of the lemma. ◀

We will also use the framework of de Berg et al. [6] for the design of subexponential-time algorithms for geometric intersection graphs. First, let us introduce some terminology. For a graph $G = (V, E)$ and $\kappa \in \mathbb{N}$, a κ -*partition* of G is a partition (P_1, \dots, P_ν) of V such that every P_i induces a connected subgraph which is a union of at most κ cliques. For a κ -partition \mathcal{P} of G , the \mathcal{P} -*contraction* of G , denoted by $G_{\mathcal{P}}$, is the graph obtained by contracting every P_i into a single vertex, that is, $V(G_{\mathcal{P}}) = \{P_1, \dots, P_\nu\}$ and $E(G_{\mathcal{P}}) = \{P_i P_j \mid \exists v_i \in P_i, v_j \in P_j: v_i v_j \in E(G)\}$. Let $\gamma: \mathbb{N} \rightarrow \mathbb{N}$ be a weight function. For a tree decomposition (T, σ) of $G_{\mathcal{P}}$, its *weighted width* with respect to γ is defined by $\max_t \sum_{P_i \in \sigma(t)} \gamma(|P_i|)$, where the maximum is over the nodes t of T .

² The constant depends on Lemma 5.

The main technical step of the algorithmic framework of de Berg et al. is the following theorem, restricted to the case of unit disk graphs.

► **Theorem 7** ([6], Theorem 2.11 applied to unit disk graphs). *For a weight function γ such that $\gamma(t) \in O(t^{1/2-\varepsilon})$ for $\varepsilon > 0$, there exists a κ -partition \mathcal{P} for $\kappa \in O(1)$ such that $G_{\mathcal{P}}$ has weighted treewidth $O(\sqrt{n})$ that can be computed in $2^{O(\sqrt{n})}$ time.*

As in Berg et al. [6], we will apply Theorem 7 with $\gamma(t) = O(\log t)$. To design a $2^{O(\sqrt{n})}$ -time algorithm, one essentially needs to show that there are $|P|^{O(1)}$ possibilities for each partition class $P \in \mathcal{P}$. We obtain this polynomial bound from Lemma 6. Specifically, let (C_1, \dots, C_ℓ) be an optimal solution satisfying the condition of Theorem 4. For $P \in \mathcal{P}$, let $\mathcal{R}(P)$ be the collection of cliques returned by Lemma 6, applied to the subset $P \subseteq V(G)$. By the lemma, for every $i \in [\ell]$, $P \cap C_i \in \mathcal{R}(P)$. On the other hand, every clique is contained in a 2×4 rectangle, therefore by Lemma 5 only constantly many cliques from C_1, \dots, C_ℓ intersect this clique. Since P is covered by at most κ cliques, it also holds that only constantly many cliques from C_1, \dots, C_ℓ intersect P , where the constant depends on κ and Lemma 5; denote this constant by λ . Later in the algorithm, we will characterize the solution (C_1, \dots, C_ℓ) on P by listing the λ cliques from $\mathcal{R}(P)$ that result from intersecting (C_1, \dots, C_ℓ) with P . We now proceed to the proof of the theorem.

Proof of Theorem 1. We first apply Theorem 7 with $\gamma(t) = \varepsilon \log t + 1$ for a sufficiently small constant $\varepsilon > 0$, obtaining a κ -partition \mathcal{P} of G , and a tree decomposition of $G_{\mathcal{P}}$ of weight at most $O(\sqrt{n})$. For $P \in \mathcal{P}$, let $\mathcal{R}(P)$ be a collection of relevant cliques in P as per Lemma 6. We define a *configuration* of P by a pair (\mathcal{C}, χ) as follows. The first element, $\mathcal{C} \subseteq \mathcal{R}(P)$, is a collection of at most λ cliques such that $\bigcup \mathcal{C} = P$. The second element, $\chi: \mathcal{C} \rightarrow \{0, 1\}$, is a mapping, which we will use to indicate whether a clique $C \in \mathcal{C}$ has been covered. We denote the set of configurations of P by Γ_P . Since $|\mathcal{R}(P)| \in |P|^{O(1)}$ by Lemma 6, there are at most $\lambda \cdot |\mathcal{R}(P)|^\lambda \cdot 2^\lambda \in |P|^{O(1)}$ many configurations. Thus, for a bag t , the number of all combinations of configurations of nodes in t is at most

$$\prod_{P \in \sigma(t)} |P|^{O(1)} = \exp \left(c \sum_{P \in \sigma(t)} \log |P| \right) \in 2^{O(\sqrt{n})}.$$

Here, c is a constant, and the second equality is due to the fact that the weighted treewidth is $O(\sqrt{n})$. The running time will be dominated by this factor.

Our dynamic programming constructs a table c_t for a bag t indexed by a configuration for each $P \in \sigma(t)$ and an integer ℓ . We describe the configuration by a mapping f that maps $P \in \sigma(t)$ to one of its configurations in Γ_P . We use the notation $f(P) = (f_{\mathcal{C}}(P), f_{\chi}(P))$. The table c_t stores Boolean values, where the entry $c_t[f, \ell]$ is true if and only if there is a collection (C_1, \dots, C_ℓ) of ℓ cliques such that

- $\bigcup_{i \in [\ell]} C_i$ covers all vertices appearing strictly below t (i.e., every vertex in $P \in \mathcal{P} \setminus \sigma(t)$ such that P appears in the subtree rooted at t is covered by $\bigcup_{i \in [\ell]} C_i$)
- $\bigcup_{i \in [\ell]} C_i$ covers all cliques $C \in f_{\mathcal{C}}(P)$ with $P \in \sigma(t)$ and $f_{\chi}(P)(C) = 1$, and
- every clique C_i , $i \in [\ell]$, contains a vertex appearing strictly below t .

Our dynamic programming will maintain this invariant.

Now we describe our dynamic programming procedure over a nice tree decomposition (see Section 2 for the definition). It follows from our invariants that the input graph admits a clique cover of size ℓ if and only if $c_r[f, \ell] = 1$ for the root r . For a non-leaf node t , we will denote its children by t', t'' (t' if t has one child).

10:8 Subexponential Algorithms for Clique Cover on Unit Disk and Unit Ball Graphs

Leaf node. Suppose that t is a leaf node, i.e., $\sigma(t) = \emptyset$. Then, $c_t[f, \ell]$ is true if and only if $\ell = 0$.

Introduce node. Suppose that t is an introduce node, i.e., $\sigma(t) = \sigma(t') \cup \{P\}$.

$$c_t[f, \ell] = \begin{cases} c_{t'}[f|_{\sigma(t')}, \ell] & \text{if } f_\chi(P)(C) = 0 \text{ for every } C \in f_C(P), \\ \text{false} & \text{otherwise.} \end{cases}$$

Here, $f|_{\sigma(t')}$ denotes the restriction of f to $\sigma(t')$. As we only consider cliques that intersect a node strictly below t , we set the table entry to false if $f_\chi(P)$ is not uniformly zero.

Forget node. Suppose that t is a forget node, i.e., $\sigma(t) = \sigma(t') \setminus \{P\}$. We have the following recurrence:

$$c_t[f, \ell] = \bigvee_{\ell' \in \{0, \dots, \ell\}, f'} c_{t'}[f', \ell']$$

where \bigvee ranges over all f' and ℓ' that satisfy the following condition. Let $H_{f'}$ be an auxiliary graph as follows. For every $C \in f'_C(P)$ with $f_\chi(P)(C) = 0$, we add a vertex h_C . Moreover, for every $P' \in \sigma(t)$ and $C \in f_C(P')$, we add a vertex h_C to H if (i) C has not been covered at t' , i.e., $f'_\chi(P')(C) = 0$ and (ii) C is covered at t , i.e., $f_\chi(P')(C) = 1$. Two vertices h_C and $h_{C'}$ are adjacent in $H_{f'}$ if and only if $C \cup C'$ is a clique in the graph G . This concludes the construction of $H_{f'}$. Note that $H_{f'}$ has size $O(\sqrt{n})$. Then \bigvee ranges over f' and ℓ' such that $H_{f'}$ has a clique cover $(D_1, \dots, D_{\ell-\ell'})$ of size $\ell - \ell'$ such that every clique D_i contains a vertex h_C for $C \in f'_C(P)$. Whether f' and ℓ' fulfills this condition can be checked in $2^{O(\sqrt{n})}$ time via dynamic programming.

Specifically, we proceed in a standard fashion for a k -COLORING/CLIQUE COVER subset-based dynamic programming. For each subset $S \subseteq V(H_{f'})$ and each integer k , $0 \leq k \leq \ell - \ell'$, we compute the Boolean value $d[S, k]$ that is equal to true if and only if the subgraph $H_{f'}[S]$ admits a clique cover of size k , where additionally every clique contains a vertex h_C for some $C \in f'_C(P)$. We initialize by setting $d[\emptyset, 0] = \text{true}$, $d[S, 0] = \text{false}$ for each $S \neq \emptyset$, and for each $S \subseteq V(H_{f'})$, $k \in [\ell - \ell']$, compute $d[S, k] = \bigvee_{D \text{ is an admissible clique in } H_{f'}[S]} d[S \setminus D, k - 1]$. Clearly, the dynamic programming table above is computed in time $2^{O(|V(H_{f'})|)} = 2^{O(\sqrt{n})}$. As there are $2^{O(\sqrt{n})}$ many choices for the configuration f' , we can compute $c_t[f, \ell]$ in overall time $2^{O(\sqrt{n})}$.

Let us verify that the invariant is maintained by the computation above. If $c_t[f, \ell]$ is set to true, then there exist f' and ℓ' satisfying the aforementioned condition, for which $c_{t'}[f', \ell']$ is also true. Since $c_{t'}[f', \ell']$ is true, there exists a collection $(C_1, \dots, C_{\ell'})$ of cliques. Also, $H_{f'}$ admits clique cover of size $\ell - \ell'$, which is also a collection of cliques in G . Combining these cliques indeed satisfies the conditions.

Join node. Suppose that t is a join node, i.e., $\sigma(t) = \sigma(t') = \sigma(t'')$. We have the recurrence:

$$c_t[f, \ell] = \bigvee_{\ell' \in \{0, \dots, \ell\}, f', f''} (c_{t'}[f', \ell'] \wedge c_{t''}[f'', \ell - \ell']),$$

where \bigvee ranges over functions f', f'' that map $P \in \sigma(t)$ to one of its configurations such that for every $P \in \sigma(t)$,

- P is partitioned in cliques in the same way, i.e., $f_C(P) = f'_C(P) = f''_C(P)$, and
- for every $C \in f_C(P)$, $f_\chi(P)(C) = 1$ if and only if C is covered in one of the children, i.e., $f'_\chi(P)(C) = 1$ or $f''_\chi(P)(C) = 1$.

To see why the invariant is maintained, note that if $c_t[f, \ell]$ is set to true, then there are ℓ' cliques certifying $c_{t'}[f', \ell']$ being true and $\ell - \ell'$ cliques certifying $c_{t''}[f'', \ell - \ell']$ being true. Putting them together, we obtain a collection of ℓ cliques satisfying the conditions.

Observe that each entry can be computed in $2^{O(\sqrt{n})}$ time. Since there are $2^{O(\sqrt{n})}$ entries, the running time is bounded by $2^{O(\sqrt{n})}$. Note that all arithmetic operations can be performed in polynomial time: we only require comparing distances between the given points and orientations between triples of given points; see the proof of Lemma 6. Theorem 7 is representation-agnostic, meaning that no additional arithmetic operations are required, except for constructing the graph from the given geometric representation.

This concludes the proof of Theorem 1. \blacktriangleleft

4 Subexponential lower bound for $d \geq 2$

In this section, we establish the impossibility of solving the CLIQUE COVER problem on d -dimensional unit ball graphs in time better than $2^{O(n^{1-1/d})}$. For this, we use the result of de Berg et al. [6], which states that, assuming ETH, GRID EMBEDDED SAT cannot be solved in time $2^{o(n)}$ time, where n is the number of variables of the given formula.

GRID EMBEDDED SAT is defined as follows. Let $G^2(n)$ denote the $n \times n$ -grid graph, where there is a vertex (i, j) for every $i, j \in [n]$ and an edge between (i, j) and (i', j') are adjacent if and only if $|i - i'| = |j - j'| = 1$. We say that a graph H is *embedded* in $G^2(n)$ if a subdivision of H is isomorphic to a subgraph of $G^2(n)$. For a CNF formula ϕ , the *incidence graph* G_ϕ of ϕ is the bipartite graph, where there is a vertex for each variable and each clause, and there is an edge between a variable vertex and a clause vertex if and only if the variable appears in the clause. A (3, 3)-CNF formula is a CNF formula where each variable appears at most 3 times and each clause has size at most 3.

GRID EMBEDDED SAT

Input: A (3, 3)-CNF formula ϕ together with an embedding of its incidence graph G_ϕ in $G^2(n)$.
Task: Is there a satisfying assignment for ϕ ?

► **Proposition 8** ([6], Theorem 3.2). *GRID EMBEDDED SAT can not be solved in time $2^{o(n)}$ unless ETH fails.*

To show ETH-hardness for \mathbb{R}^d , $d \geq 3$, we use the *cube wiring theorem* due to de Berg et al. [6]. Let $B^d(n)$ denote $[n]^d$ and $G^d(n)$ denote the d -dimensional hypercube over $B^d(n)$. Also, for $p \in B^{d-1}(n)$ and $h \in [n]$, let $\xi^h(p) = (p_1, \dots, p_{d-1}, h) \in B^d(n)$. For $s \in \mathbb{N}$, a set $P \subseteq \mathbb{Z}^{d-1}$ is said to be s -spaced if there is an integer $0 \leq r < s$ such that for every $p = (p_1, \dots, p_{d-1}) \in P$ and $i \in [d-1]$, $p_i \equiv r \pmod{s}$.

► **Theorem 9** (Cube wiring theorem [6]). *For $d \geq 3$, let P and Q be two 2-spaced subsets of $B^{d-1}(n)$ and let M be a perfect matching in the bipartite graph $(P \cup Q, P \times Q)$. Then, for $n' \in O(n)$, $G^d(n')$ contains vertex-disjoint paths that connect $\xi^1(p)$ and $\xi^{n'}(q)$ for every $p, q \in M$.*

Now we prove our theorem.

► **Theorem 2.** *Assuming the ETH, CLIQUE COVER on n -vertex unit ball graphs in \mathbb{R}^d does not admit a $2^{o(n^{1-1/d})}$ -time algorithm, for any $d > 1$, even if the geometric representation of polynomial bit-length is given in the input.*

10:10 Subexponential Algorithms for Clique Cover on Unit Disk and Unit Ball Graphs

Proof. We first present a reduction from GRID EMBEDDED SAT to CLIQUE COVER on unit disk graphs. Let ϕ be a (3,3)-CNF formula. We may assume that each variable in ϕ appears twice positively and once negatively: For every variable v where its occurrences are all positive or negative, delete the clauses containing v . Also, for every variable v appears twice negatively and once positively, flip its sign. We first describe how to construct a GRID EMBEDDED SAT instance (G, k) from ϕ , and specify the embedding later.

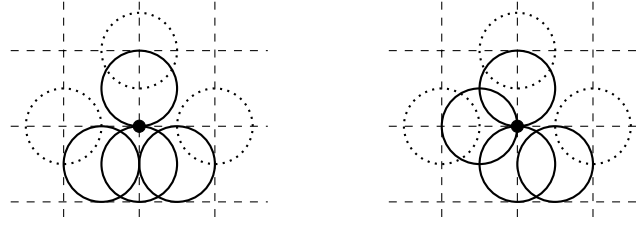
- For each variable x , we create a *variable gadget*, which is obtained by gluing K_3 and K_2 over one vertex, i.e., it is a paw, consisting of four vertices u_x, u'_x, v_x, w_x and edges $u_x u'_x, u_x v_x, u'_x v_x, v_x w_x$. We will call u_x, u'_x, w_x *connection vertices*.
- For each clause, we introduce a single vertex C . We call it a *clause gadget*.
- We construct a *wire gadget*, which will be used to connect a variable gadget to a clause gadget in the embedding. A wire corresponding to a positive literal x is a path with an even number of edges, starting at u_x or u'_x from the variable gadget of x , and ending at the corresponding clause vertex. For a negative literal, the path starts at w_x instead. We call a wire *activated* if the value of the corresponding literal of the connection vertex is true. We call a wire if the corresponding literal is set to true.

This completes the construction of G . Let L be the total edge length of all wires. We show that the formula ϕ has a satisfying assignment if and only if G has a clique cover of size $k = n + L/2$, where n is the number of variables.

Correctness. Suppose that formula ϕ has a satisfying assignment. We construct a clique cover of G as follows. For each variable x , we pick a clique $\{u_x, u'_x, v_x\}$ if x is assigned true and $\{v_x, w_x\}$ otherwise. For each wire with 2ℓ edges, pick ℓ edges as K_2 's so that all inner vertices and the connection vertex is covered if the wire is activated, and all inner vertices and the clause vertex is covered otherwise. Since the assignment satisfies all the clauses, every clause gadget has at least one activated wire. If more than one wire ends with K_2 containing a clause, then arbitrarily pick one wire and reduce the internal vertices of the remaining wires into a K_1 and pick it into the solution. Hence, all vertices in the variable, wire and clause gadget are covered. We obtain a clique partition of G with $k = n + L/2$ cliques.

Conversely, assume G has a clique cover of size k . Each variable gadget contains at least one clique that covers the common vertices of the gadget. Since in a wire of length 2ℓ , there are $2\ell - 1$ internal vertices, and only two vertices of a wire can be covered by a clique. Thus, wire gadgets contain at least $L/2$ cliques. Since $k = n + L/2$, the solution contains exactly one clique for every variable gadget and each wire of length 2ℓ will have exactly ℓ cliques. Since every clause vertex belongs to a clique in the solution, a literal exists such that the corresponding wire is activated. Then, the respective connection vertex is not a part of the wire clusters and thus is a part of the vertex cluster. We assign the variable's value based on which side the clique in each variable gadget picks. If the clique picks connection vertices corresponding to K_3 , we set the variable to be true. If the clique contains connection vertices corresponding to K_2 , then we set the variable to be false. Otherwise, we set variable values arbitrarily.

Embedding. Suppose that $d = 2$. Let \mathcal{D} be a grid embedding of G_ϕ . We start by taking a 2-refinement of \mathcal{D} . This will ensure that each wire gadget has even length. For every vertex in G_ϕ , we introduce a disk (of diameter 1) centered at its coordinate, unless it is a variable vertex. For a variable x , let (i, j) be its coordinate in \mathcal{D} . Without loss generality, assume that three vertices adjacent to x in G_ϕ are at $(i - 1, j)$, $(i, j + 1)$, and $(i + 1, j)$. There are three cases depending on which edge in G_ϕ incident with x connects to a negative literal.



■ **Figure 2** Two cases for a variable gadget. The coordinate (i, j) is marked by the black dot. The dotted disks are part of wire gadgets. Note that the variable gadget has exactly one disk intersecting a dotted disk.

First, suppose that the edge between (i, j) and $(i, j + 1)$ leads to a negative literal. Then, introduce four disks centered at $(i - 1/2, j - 1/2)$, $(i + 1/2, j - 1/2)$ (corresponding to u_x and u'_x), $(i, j - 1/2)$ (corresponding to v_x), and $(i, j + 1/2)$ (corresponding to w_x). Otherwise, suppose that the between (i, j) and $(i + 1, j)$ leads to a negative literal. Then, introduce four disks centered at $(i - 1/2, j)$, $(i, j + 1/2)$ (corresponding to u_x and u'_x), $(i, j - 1/2)$ (corresponding to v_x), and $(i + 1/2, j - 1/2)$ (corresponding to w_x). See Figure 2 for an illustration.

Note that only polynomial precision in coordinates is required to construct the instance, therefore the hardness also holds if the representation is given.

For $d \geq 3$, for every variable, we place three vertices adjacent to its variable gadget in a $(d-1)$ -hypercube of side length 3. We then place all these hypercubes into a $(d-1)$ -hypercube of side length $n^{O(\frac{1}{d-1})}$. Placing the clause gadgets on $B^{d-1}(m^{\frac{1}{d-1}})$, we apply the cube wiring theorem (Theorem 9) to obtain an embedding into $B^d(n')$ for $n' \in O(n)$. We then embed the variable gadgets similarly to the case $d = 2$. ◀

5 Exponential lower bound for $d = 5$

In this section, we present a hardness reduction excluding better-than-exponential running time for CLIQUE COVER on unit ball graphs in dimension at least 5. We restate the result next.

▶ **Theorem 3.** *Assuming the ETH, CLIQUE COVER on n -vertex unit ball graphs in \mathbb{R}^5 does not admit a $2^{o(n)}$ -time algorithm, even if the geometric representation of polynomial bit-length is given in the input.*

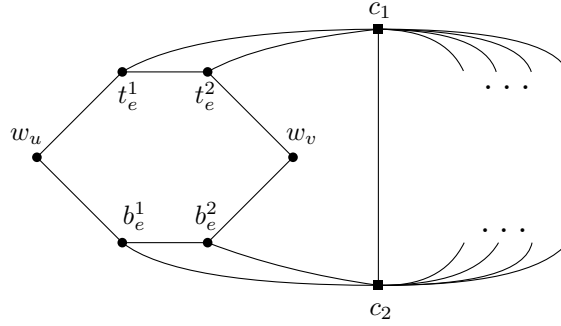
Proof. We show a reduction from 3-COLORING to CLIQUE COVER, where the target instance is a unit ball graph in \mathbb{R}^d . Let G be the graph in the instance of 3-COLORING. We first construct an enhanced graph G' from G and argue that this makes an equivalent instance of 3-COLORING. Then, we show that the complement of the enhanced graph G' admits a unit ball representation in \mathbb{R}^5 . Since solving 3-COLORING on G' is equivalent to solving 3-CLIQUE COVER on $\overline{G'}$, and 3-CLIQUE COVER on unit ball graphs in \mathbb{R}^5 is the special case of CLIQUE COVER with $k = 3$ on the same class of graphs, this completes the reduction.

We now move to the details. First, we define the enhanced graph G' . The vertex set of G' contains one vertex for each vertex of G , four vertices for each edge of G , and two additional special vertices. Formally, $V(G') = W \cup T \cup B \cup C$, where $W = \{w_v : v \in V(G)\}$, $T = \{t_e^1, t_e^2 : e \in E(G)\}$, $B = \{b_e^1, b_e^2 : e \in E(G)\}$, $C = \{c_1, c_2\}$. The edges are as follows: for every edge $e = uv \in E(G)$, we construct $w_u t_e^1$, $t_e^1 t_e^2$, $t_e^2 w_v$, and $w_u b_e^1$, $b_e^1 b_e^2$, $b_e^2 w_v$. Additionally, c_1 is adjacent to all vertices of T , c_2 is adjacent to all vertices of B , and c_1 and c_2 are adjacent. Formally, the edge set of G' is

10:12 Subexponential Algorithms for Clique Cover on Unit Disk and Unit Ball Graphs

$$E(G') = \{w_u t_e^1, t_e^1 t_e^2, t_e^2 w_v, w_u b_e^1, b_e^1 t_e^2, b_e^2 w_v, t_e^1 c_1, t_e^2 c_1, b_e^1 c_2, b_e^2 c_2 : e \in E(G)\} \cup \{c_1 c_2\}.$$

Intuitively, G' is obtained from G by replacing each edge $e \in E(G)$ with two copies of its 2-subdivision: the vertices t_e^1 and t_e^2 are internal vertices of the first copy, and the vertices b_e^1 and b_e^2 are internal vertices of the second copy. Moreover, there are two special vertices c_1 and c_2 that are adjacent to each other, and c_1 is adjacent to the internal vertices of the first 2-subdivision, while c_2 is adjacent to internal vertices of the second 2-subdivision. See Figure 3 for an illustration of the edge gadget.



■ **Figure 3** Edge gadget in G' , encoding the edge e between vertices u, v in G . Vertices c_1 and c_2 are connected in the same way to all edge gadgets.

We now argue that G' is equivalent to G in terms of 3-colorings.

▷ **Claim 10.** G admits a 3-coloring if and only if G' admits a 3-coloring.

Proof. Let $c : V(G) \rightarrow \{1, 2, 3\}$ be the 3-coloring of G , we construct a 3-coloring c' of G' . Let c' coincide with c on the vertices of W ; let $c'(c_1) = 1$ and $c'(c_2) = 2$. We now assign colors to vertices t_e^h and b_e^h for $e \in E(G), h \in [2]$.

Consider an edge $e = uv \in E(G)$, so that u is adjacent to t_e^1 and b_e^1 in G' . The vertex t_e^1 has an available color since only c_1 and u have assigned colors among its neighbors; assign this color to t_e^1 . Now, assume there is no available color for t_e^2 , therefore all three colors appear among c_1, v, t_e^1 . Since $c'(c_1) = 1$, either $c'(v) = 2$ and $c'(t_e^1) = 3$, or the other way around. In the former case, $c'(u) \neq 2$ since $c(\cdot)$ is a proper 3-coloring of G . Assign $c'(t_e^1) = 2$ and $c'(t_e^2) = 3$; all edges between the considered vertices are properly colored. In the alternative case, the argument is symmetric: $c'(v) = 3$ and $c'(u) \neq 3$; assign $c'(t_e^1) = 3$ and $c'(t_e^2)$. The argument for the vertices b_e^1 and b_e^2 is analogous.

In the other direction, consider a 3-coloring c' of G' ; we claim that the restriction c of c' to $V(G)$ is a proper 3-coloring of G . Assume this is not the case, therefore there exists an edge $e = uv \in E(G)$ with $c'(u) = c(u) = c(v) = c'(v)$. Since c_1 and c_2 are adjacent in G' , they receive different colors under c' and so either $c'(c_1) \neq c'(u)$ or $c'(c_2) \neq c'(u)$; w.l.o.g. assume the former case. The vertex t_e^1 has only one available color since it cannot coincide with $c'(u)$ and $c'(c_1)$, which are two distinct colors. Then the neighborhood of t_e^2 contains all three colors, since $c'(u) = c'(v)$. This contradicts the fact that t_e^2 is properly colored by c' . ◁

Then we proceed to construct a unit ball representation of the complement of G' in \mathbb{R}^5 . To this end, we describe the locations of all vertices in G' under the embedding, and argue that the distance between the locations exceeds a certain value if and only if the respective pair of vertices is adjacent in G' .

First, we embed the vertices of T , B in C in the first three dimensions, i.e., their images are always zero in coordinates 4 and 5. Then, we embed the vertices of W in the other two dimensions, i.e., such the coordinates 1–3 are zeroed out. Finally, we shift the embedding of T and B slightly to achieve the desired edges between W and $T \cup B$.

Let $\epsilon > 0$ be a constant to be defined later. We place c_1 and c_2 symmetrically across the origin at distance of $\sqrt{3} - \sqrt{2}/2 + \epsilon$ along the first coordinate; that is,

$$\begin{aligned}\pi(c_1) &= (\sqrt{3} - \sqrt{2}/2 + \epsilon, 0, 0, 0, 0), \\ \pi(c_2) &= (-\sqrt{3} + \sqrt{2}/2 - \epsilon, 0, 0, 0, 0).\end{aligned}$$

We then position the set T on the circumference of a circle with the center on the Ox_1 axis lying in the plane orthogonal to the axis, with radius $r = 1 + \epsilon'$, and such that its center is $\sqrt{2}/2 - \epsilon$ away from the origin towards $-\infty$. We shall define the precise value of ϵ' later. The points of T_1 occupy the “top cap” of the circumference, i.e., a small arc close to $x_2 = r$, and the points of T_2 occupy the “bottom cap”, i.e., close to $x_2 = -r$. We aim that for each $e \in E(G)$, t_e^1 lies directly opposite to t_e^2 , while the remaining points are sufficiently close to each of them. Let $E(G) = \{e_1, \dots, e_m\}$, we position the points $t_{e_1}^1, \dots, t_{e_m}^1$ evenly along the arc starting from the “top” of the circle, such that the angle between the two consecutive points is always δ/m , measured from the center of the circle. We then place the points $t_{e_1}^2, \dots, t_{e_m}^2$ similarly, directly opposite to their counterparts. We define the exact positions as follows:

$$\begin{aligned}\pi(t_{e_j}^1) &= (-\sqrt{2}/2 + \epsilon, r \cdot \cos(\delta \cdot j/m), r \cdot \sin(\delta \cdot j/m), 0, 0), \\ \pi(t_{e_j}^2) &= (-\sqrt{2}/2 + \epsilon, -r \cdot \cos(\delta \cdot j/m), -r \cdot \sin(\delta \cdot j/m), 0, 0).\end{aligned}$$

The points of B are positioned very similarly, except that they are placed in a circle placed opposite across the origin to the circle above, i.e., its center is the point $(\sqrt{2}/2, 0, 0, 0, 0)$. And the points of B_1 (B_2) are placed close to $x_3 = r$ ($x_3 = -r$). Formally,

$$\begin{aligned}\pi(b_{e_j}^1) &= (\sqrt{2}/2 - \epsilon, -r \cdot \sin(\delta \cdot j/m), r \cdot \cos(\delta \cdot j/m), 0, 0), \\ \pi(b_{e_j}^2) &= (\sqrt{2}/2 - \epsilon, r \cdot \sin(\delta \cdot j/m), -r \cdot \cos(\delta \cdot j/m), 0, 0).\end{aligned}$$

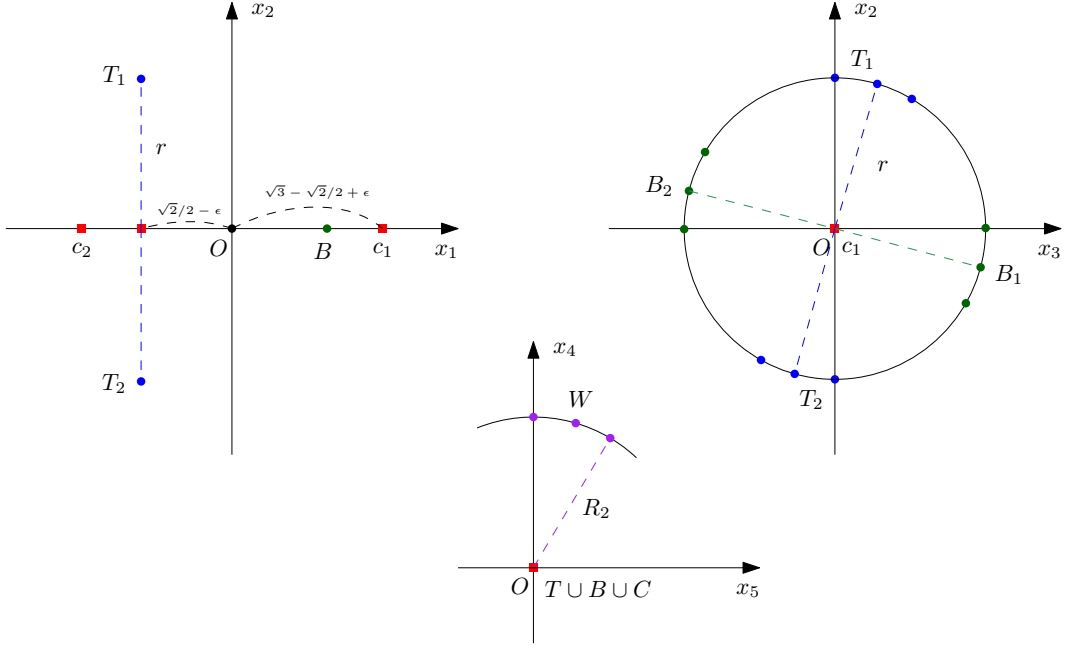
Note that the image of every point in $T \cup B$ is exactly $R_1 = \sqrt{(\sqrt{2}/2 - \epsilon)^2 + r^2}$ away from the origin. Assume ϵ is such that $R_1 < 4$, and let $R_2 = \sqrt{4 - R_1^2}$. We place the points of W in the plane Ox_4x_5 exactly at the distance of R_2 from the origin. Namely, consider the circle in Ox_4x_5 centered at the origin with the radius of R_2 . We place the points of $W = \{w_1, \dots, w_n\}$ evenly along the circumference, such that the angle between consecutive points is exactly δ/n :

$$\pi(w_i) = (0, 0, 0, R_2 \cdot \cos(\delta \cdot i/n), R_2 \cdot \sin(\delta \cdot i/n)).$$

See Figure 4 for the illustration of the embedding π .

We now show that π “nearly” gives the desired embedding of G' . That is, we show that every adjacent pair is at distance strictly more than 2 and every non-adjacent pair is at distance strictly less than 2, except for the pairs of form (w, v) , $w \in W$, $v \in T \cup B$, which are at distance exactly 2. Later we will slightly modify the embedding π to make sure that exactly the required pairs of this form are sufficiently far from each other.

10:14 Subexponential Algorithms for Clique Cover on Unit Disk and Unit Ball Graphs



■ **Figure 4** Illustration of the embedding π , showed by schematic projections on the three planes.

▷ **Claim 11.** There exists $\xi > 0$ such that the following holds:

$$\|\pi(w) - \pi(v)\| = 2, \text{ for each } w \in W, v \in T \cup B, \quad (1)$$

$$\|\pi(t_e^1) - \pi(t_e^2)\| \geq 2 + \xi, \text{ for each } e \in E(G), \quad (2)$$

$$\|\pi(b_e^1) - \pi(b_e^2)\| \geq 2 + \xi, \text{ for each } e \in E(G), \quad (3)$$

$$\|\pi(t_e^h) - \pi(c_1)\| \geq 2 + \xi, \text{ for each } e \in E(G), h \in [2], \quad (4)$$

$$\|\pi(b_e^h) - \pi(c_2)\| \geq 2 + \xi, \text{ for each } e \in E(G), h \in [2], \quad (5)$$

$$\|\pi(c_1) - \pi(c_2)\| \geq 2 + \xi, \quad (6)$$

and for any other two vertices v, u of G' , the distance $\|\pi(v) - \pi(u)\|$ is at most $2 - \xi$.

Proof. Let $\xi = \epsilon'/2$. Equation (1) holds immediately by construction, since each $v \in T \cup B$ is situated exactly R_1 away from the origin, each $w \in W$ exactly R_2 away from the origin, $T \cup B$ is contained in the 3-dimensional subspace $Ox_1x_2x_3$ which is orthogonal to the plane Ox_4x_5 where W is contained, and $R_1^2 + R_2^2 = 4$ by definition of R_2 .

For Equation (2), observe that $\|\pi(t_e^1) - \pi(t_e^2)\| = 2 + 2\epsilon'$ for each $e \in E(G)$ since these two points are situated diametrically opposite to each other on a circle of radius $1 + \epsilon'$. Therefore, $\|\pi(t_e^1) - \pi(t_e^2)\| \geq 2 + \xi$ since $\epsilon' = 2\xi \geq \xi/2$. On the other hand, consider the points t_e^1 and $t_{e'}^2$, for $e \neq e'$. Since $t_{e'}^2$ lies on the same circle at the angle of at least δ/m away from t_e^2 , the distance between t_e^1 and $t_{e'}^2$ is at most $2(1 + \epsilon') \cos(\delta/2m)$. Therefore, if it holds that $(1 + \epsilon') \cos(\delta/2m) \leq 1 + \xi/2 = 1 + \epsilon'/4$, then all distances between the points of T are as desired. Moreover, exactly the same arguments hold for Equation (3) and the distances between the points of B . We now show this bound given that ϵ' is sufficiently small:

$$\begin{aligned} \epsilon' \leq \frac{\delta^2}{20m^2} &\implies \frac{1 + \epsilon'}{\epsilon'} \geq \frac{20m^2}{\delta^2} \implies \frac{\delta^2}{16m^2} \geq \frac{5}{4} \cdot \frac{\epsilon'}{1 + \epsilon'} \\ &\implies 1 - \frac{\delta^2}{16m^2} \leq 1 - \frac{5}{4} \cdot \frac{\epsilon'}{1 + \epsilon'} = \frac{1 - \epsilon'/4}{1 + \epsilon'} \\ &\implies (1 + \epsilon') \cdot \cos \frac{\delta}{2m} \leq (1 + \epsilon') \cdot \left(1 - \frac{\delta^2}{16m^2}\right) \leq 1 - \epsilon'/4. \end{aligned}$$

Here, we also use that δ is a sufficiently small constant.

Consider now Equation (4), the distance between $\pi(c_1)$ and $\pi(t_e^h)$ is equal to $\sqrt{3+r^2}$ for each $e \in E(G)$ and $h \in [2]$. It is therefore sufficient to have $\sqrt{3+(1+\epsilon')^2} \geq 2+\xi$, which holds since $\epsilon' = 2\xi$, and the same argument holds for Equation (5) because of the symmetry. Note that the distance between $\pi(c_2)$ and $\pi(t_e^h)$ for any $e \in E(G)$, $h \in [2]$ is equal to

$$\sqrt{(\sqrt{3}-\sqrt{2}+2\epsilon)^2+r^2} = \sqrt{(\sqrt{3}-\sqrt{2})^2+1+O(\epsilon+\epsilon')} < 1.5 \leq 2-\xi,$$

for sufficiently small ϵ , ϵ' and ξ . The same holds for $\pi(c_1)$ and $\pi(b_e^h)$ for any $e \in E(G)$, $h \in [2]$.

For Equation (6),

$$\|\pi(c_1) - \pi(c_2)\| = 2\sqrt{3} - \sqrt{2} + 2\epsilon \geq 2.049 \geq 2 + \xi$$

when ξ is sufficiently small.

It remains to verify that pairwise distances not discussed above are bounded by $2-\xi$. Consider first $w \in W$ and c_h for $h \in [2]$, the respective squared distance is

$$\begin{aligned} \|\pi(w) - \pi(c_h)\|^2 &= R_2^2 + (\sqrt{3} - \sqrt{2}/2 + \epsilon)^2 = 4 - R_1^2 + (\sqrt{3} - \sqrt{2}/2)^2 + O(\epsilon) \\ &\leq 5.025 - R_1^2 + O(\epsilon) = 5.025 - (\sqrt{2}/2 - \epsilon)^2 - (1 + \epsilon')^2 + O(\epsilon) \\ &= 5.025 - 1/2 - 1 + O(\epsilon + \epsilon') = 3.525 + O(\epsilon + \epsilon') \leq 3.8 \end{aligned}$$

for sufficiently small ϵ and ϵ' . Therefore, $\|\pi(w) - \pi(c_h)\| \leq 2-\xi$ when ξ is sufficiently small.

Note also that when δ is a sufficiently small constant, distances between all pairs of vertices in W under π are at most 1, since the images occupy an arc which is a small fraction of a constant-radius circle, and the same holds for pairs in T_1, T_2, B_1, B_2 .

Finally, it remains to consider pairs of the form $t \in T$, $b \in B$. Observe that when projecting T and B orthogonally on the plane Ox_2x_3 , these sets lie on the same circle of radius $r = 1 + \epsilon'$, and the radial distance between a point in T and a point in B is always at most $\pi/2 + \delta$, since T_1, B_1, T_2, B_2 are each rotated $\pi/2$ further away from the previous set, and each of the four sets occupies an arc of radial length at most δ . Therefore,

$$\begin{aligned} (2-\xi)^2 - \|\pi(t) - \pi(b)\|^2 &\geq (2-\xi)^2 - (\sqrt{2}-2\epsilon)^2 - 2(1+\epsilon')^2(1+\sin\delta) \\ &\geq 4(\sqrt{2}-1)\epsilon - 4\xi - 6\epsilon' - 8\delta, \end{aligned}$$

by using $\epsilon', \epsilon \leq 1$ and $\sin\delta \leq \delta$. The above value is greater than zero if $\epsilon \geq 8(\epsilon' + \xi + \delta)$.

To conclude the proof of the claim, we note that the parameters $\xi, \epsilon', \delta, \epsilon$ clearly admit values that satisfy all the restrictions above. Indeed, it is only required that each of them does not exceed a certain constant independent of the other parameters, and additionally that $2\xi = \epsilon' \leq \frac{\delta^2}{20m^2}$, and $\epsilon \geq 8(\epsilon' + \xi + \delta)$. \square

Finally, we construct the embedding π' that gives the desired representation of G' . For that, we modify π in the following way: we only change the images of vertices in $T \cup B$. Namely, $\pi'(v) = \pi(v)$ for $v \in V(G') \setminus (T \cup B)$, and for each $e = uv \in E(G)$,

$$\begin{aligned} \pi'(t_e^1) &= \pi(t_e^1) + \theta \cdot \overrightarrow{\pi(u)O}, \\ \pi'(b_e^1) &= \pi(b_e^1) + \theta \cdot \overrightarrow{\pi(u)O}, \\ \pi'(t_e^2) &= \pi(t_e^2) + \theta \cdot \overrightarrow{\pi(v)O}, \\ \pi'(b_e^2) &= \pi(b_e^2) + \theta \cdot \overrightarrow{\pi(v)O}, \end{aligned}$$

where $\theta > 0$ is a small value to be defined later. We show that the embedding π' is indeed a unit ball representation of the complement of G' .

10:16 Subexponential Algorithms for Clique Cover on Unit Disk and Unit Ball Graphs

▷ **Claim 12.** For every $u, v \in V(G')$, $uv \in E(G')$ if and only if $\|\pi'(u) - \pi'(v)\| > D$, for some $D > 2$.

Proof. First, we consider distances between the pairs $\pi'(w), \pi'(v)$, where $w \in W$, $v \in T \cup B$. Let $v \in T \cup B$ and let w be the unique vertex in W such that $\pi'(v) = \pi(v) + \theta \cdot \overrightarrow{\pi(w)O}$. The respective squared distance is then $\|\pi'(v) - \pi'(w)\|^2 = R_1^2 + (R_2 + \theta)^2$. On the other hand, consider a vertex $w' \in W$ with $w' \neq w$. For v and w' , the squared distance is $\|\pi'(v) - \pi'(w')\|^2 \leq R_1^2 + R_2^2 + \theta^2 + 2R_2\theta \cos(\delta/n)$, since the distance is independent in $Ox_1x_2x_3$ and Ox_4x_5 , and in the latter plane the vertex w' is at least at angle of δ/n away from the line $\overrightarrow{O\pi(w)}$, which by the law of cosines gives the upper bound above.

By setting $D = \sqrt{R_1^2 + R_2^2 + \theta^2 + 2R_2\theta \cos(\delta/n)}$ we therefore achieve that $\|w' - v\| \leq D$ for each $w' \neq w$, while $\|w - v\| > D$ since

$$(R_1^2 + (R_2 + \theta)^2) - (R_1^2 + R_2^2 + \theta^2 + 2R_2\theta \cos(\delta/n)) = 2R_2\theta \cdot (1 - \cos(\delta/n)) > 0.$$

Observe that $2 < D$ since $R_1^2 + R_2^2 = 4$, and that $D < 2 + \xi/2$ for a sufficiently small value of θ ; fix θ so that the latter holds. We now verify the distance condition for the remaining pairs. First, the distance between vertices of W and c_1/c_2 is the same under π and π' , so it is at most $2 < D$. It remains to consider distances between pairs of vertices in X , where $X = T \cup B \cup C$. For each $v, u \in X$,

$$\|\pi'(u) - \pi'(v)\|^2 - \|\pi(u) - \pi(v)\|^2 \leq 2\theta^2 R_2^2$$

since the change from π to π' shifts each vertex by at most the vector of $\theta \cdot \overrightarrow{\pi(w)O}$ for some $w \in W$; the length of this vector is θR_2 , and π acts only into the subspace $Ox_1x_2x_3$ on X . Clearly, for sufficiently small θ we get that

$$-\xi/2 < \|\pi'(u) - \pi'(v)\| - \|\pi(u) - \pi(v)\| < \xi/2.$$

Therefore, all distances that were at most $2 - \xi$ (at least $2 + \xi$, respectively) under π from Claim 11 remain at most $2 - \xi/2$ (at least $2 + \xi/2$, respectively) under π' . Since $2 - \xi/2 < 2 < D < 2 + \xi/2$, the proof of the claim is concluded. ◁

By Claim 12, we get the correctness of the presented reduction. It remains to observe that the reduction can be done in polynomial time: Only precision polynomial in input size is required for the parameters used for the computation of the coordinates. Since in the resulting instance of CLIQUE COVER there are $O(n + m)$ vertices, and 3-COLORING does not admit a $2^{o(n+m)}$ -time algorithm under the ETH, the statement of the theorem follows. ◀

References


- 1 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.
- 2 Marthe Bonamy, Édouard Bonnet, Nicolas Bousquet, Pierre Charbit, Panos Giannopoulos, Eun Jung Kim, Pawel Rzazewski, Florian Sikora, and Stéphan Thomassé. EPTAS and subexponential algorithm for maximum clique on disk and unit ball graphs. *J. ACM*, 68(2):9:1–9:38, 2021. doi:10.1145/3433160.
- 3 Heinz Breu and David G. Kirkpatrick. Unit disk graph recognition is np-hard. *Computational Geometry*, 9(1):3–24, 1998. Special Issue on Geometric Representations of Graphs. doi:10.1016/S0925-7721(97)00014-X.
- 4 Vasilis Capovleas, Günter Rote, and Gerhard J. Woeginger. Geometric clusterings. *J. Algorithms*, 12(2):341–356, 1991. doi:10.1016/0196-6774(91)90007-L.

- 5 Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discret. Math.*, 86(1-3):165–177, 1990. doi:10.1016/0012-365X(90)90358-0.
- 6 Mark de Berg, Hans L. Bodlaender, Sándor Kisfaludi-Bak, Dániel Marx, and Tom C. van der Zanden. A framework for exponential-time-hypothesis-tight algorithms and lower bounds in geometric intersection graphs. *SIAM J. Comput.*, 49(6):1291–1331, 2020. doi:10.1137/20M1320870.
- 7 Adrian Dumitrescu and János Pach. Minimum clique partition in unit disk graphs. *Graphs Comb.*, 27(3):399–411, 2011. doi:10.1007/s00373-011-1026-1.
- 8 Venkatesan Guruswami and Sanjeev Khanna. On the hardness of 4-coloring a 3-colorable graph. *SIAM J. Discret. Math.*, 18(1):30–40, 2004. doi:10.1137/S0895480100376794.
- 9 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k -SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.
- 10 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.
- 11 Ross J. Kang and Tobias Müller. Sphere and dot product representations of graphs. *Discret. Comput. Geom.*, 47(3):548–568, 2012. doi:10.1007/S00454-012-9394-8.
- 12 Sanjeev Khanna, Nathan Linial, and Shmuel Safra. On the hardness of approximating the chromatic number. *Comb.*, 20(3):393–415, 2000. doi:10.1007/S004930070013.
- 13 Imran A. Pirwani and Mohammad R. Salavatipour. A weakly robust PTAS for minimum clique partition in unit disk graphs. *Algorithmica*, 62(3-4):1050–1072, 2012. doi:10.1007/S00453-011-9503-8.

Solving Co-Path/Cycle Packing and Co-Path Packing Faster Than 3^k

Yuxi Liu ✉

University of Electronic Science and Technology of China, Chengdu, China

Mingyu Xiao ✉ 

University of Electronic Science and Technology of China, Chengdu, China

Abstract

The CO-PATH/CYCLE PACKING problem (resp. The CO-PATH PACKING problem) asks whether we can delete at most k vertices from the input graph such that the remaining graph is a collection of induced paths and cycles (resp. induced paths). These two problems are fundamental graph problems that have important applications in bioinformatics. Although these two problems have been extensively studied in parameterized algorithms, it seems hard to break the running time bound 3^k . In 2015, Feng et al. provided an $O^*(3^k)$ -time randomized algorithms for both of them. Recently, Tsur showed that they can be solved in $O^*(3^k)$ time deterministically. In this paper, by combining several techniques such as path decomposition, dynamic programming, cut & count, and branch-and-search methods, we show that CO-PATH/CYCLE PACKING can be solved in $O^*(2.8192^k)$ time deterministically and CO-PATH PACKING can be solved in $O^*(2.9241^k)$ time with failure probability $\leq 1/3$. As a by-product, we also show that the CO-PATH PACKING problem can be solved in $O^*(5^p)$ time with probability at least $2/3$ if a path decomposition of width p is given.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Graph Algorithms, Parameterized Algorithms, Co-Path/Cycle Packing, Co-Path Packing, Cut & Count, Path Decomposition

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.11

Related Version *Full Version:* <https://arxiv.org/abs/2406.10829> [15]

Funding The work is supported by the National Natural Science Foundation of China, under the grants 62372095, 62172077, and 62350710215.

1 Introduction

In the classic VERTEX COVER problem, the input is a graph G and an integer k , and the problem asks whether it is possible to delete at most k vertices such that the maximum degree of the remaining graph is at most 0. A natural generalization of VERTEX COVER is that: can we delete at most k vertices such that the maximum degree of the remaining graph is at most d ? Formally, for every integer $d \geq 0$, we consider the following d -BOUNDED-DEGREE VERTEX DELETION problem.

d -BOUNDED-DEGREE VERTEX DELETION

Instance: A graph $G = (V, E)$ and two integers d and k .

Question: Is there a set of at most k vertices whose removal from G results in a graph with maximum degree at most d ?



© Yuxi Liu and Mingyu Xiao;

licensed under Creative Commons License CC-BY 4.0

19th International Symposium on Parameterized and Exact Computation (IPEC 2024).

Editors: Édouard Bonnet and Paweł Rzażewski; Article No. 11; pp. 11:1–11:17

Leibniz International Proceedings in Informatics



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

The d -BOUNDED-DEGREE VERTEX DELETION problem finds applications in computational biology [7] and social network analysis [16]. In this paper, we focus on the case that $d = 2$, which is referred to as the CO-PATH/CYCLE PACKING problem. The CO-PATH/CYCLE PACKING problem also has many applications in computational biology [4]. Formally, the CO-PATH/CYCLE PACKING problem is defined as follows.

CO-PATH/CYCLE PACKING

Instance: A graph $G = (V, E)$ and an integer k .

Question: Is there a vertex subset $S \subseteq V$ of size at most k whose deletion makes the graph a collection of induced paths and cycles?

We also focus on a similar problem called CO-PATH PACKING, which allows only paths in the remaining graph, defined as follows.

CO-PATH PACKING

Instance: A graph $G = (V, E)$ and an integer k .

Question: Is there a vertex subset $S \subseteq V$ of size at most k whose deletion makes the graph a collection of induced paths?

Related Work. In this paper, we mainly consider parameterized algorithms. When d is an input, the general d -BOUNDED-DEGREE VERTEX DELETION problem is W[2]-hard with the parameter k [7]. Xiao [22] gave a deterministic algorithm that solves d -BOUNDED-DEGREE VERTEX DELETION in $O^*((d+1)^k)$ time for every $d \geq 3$, which implies that the problem is FPT with parameter $k+d$. In term of treewidth (tw), van Rooij [21] gave an $O^*((d+2)^{tw})$ -time algorithm to solve d -BOUNDED-DEGREE VERTEX DELETION for every $d \geq 1$. Lampis and Vasilakis [14] showed that no algorithm can solve d -BOUNDED-DEGREE VERTEX DELETION in time $(d+2-\epsilon)^{tw}n^{O(1)}$, for any $\epsilon > 0$ and for any fixed $d \geq 1$ unless the SETH is false. The upper and lower bounds have matched. This problem has also been extensively studied in kernelization. Fellows et al. [7] and Xiao [23] gave a generated form of the NT-theorem that study polynomial kernels for the problem with each fixed d .

For each fixed small d , d -BOUNDED-DEGREE VERTEX DELETION has also been paid certain attention. The 0-BOUNDED-DEGREE VERTEX DELETION problem is referred to as VERTEX COVER, which is one of the most fundamental problems in parameterized algorithms. For a long period of time, the algorithm of Chen et al. [3] held the best-known running time of $O^*(1.2738^k)$, and recently this result was improved by Harris and Narayanaswamy [11] to $O^*(1.25284^k)$. The 1-BOUNDED-DEGREE VERTEX DELETION problem is referred to as P_3 VERTEX COVER, where Tu [20] achieved a running time of $O^*(2^k)$ by using iterative compression. This result was later improved by Katrenič [12] to $O^*(1.8127^k)$. Then, Chang et. al. [2] gave an $O^*(1.7964^k)$ -time polynomial-space algorithm and an $O^*(1.7485^k)$ -time exponential-space algorithm. Xiao and Kou [24] gave an $O^*(1.7485^k)$ -time polynomial-space algorithm. This result was improved by Tsur [18] to $O^*(1.713^k)$ through a branch-and-search approach and finally by Červený and Suchý [1] to $O^*(1.708^k)$ through using an automated framework for generating parameterized branching algorithms.

CO-PATH/CYCLE PACKING is the special case of d -BOUNDED-DEGREE VERTEX DELETION with $d = 2$. A closely related problem is CO-PATH PACKING, where even cycles are not allowed in the remaining graph. Chen et al. [4] initially showed that CO-PATH/CYCLE PACKING and CO-PATH PACKING can be solved in $O^*(3.24^k)$ time, and a finding subsequently

■ **Table 1** Algorithms for CO-PATH/CYCLE PACKING and CO-PATH PACKING.

Years	References	CO-PATH/CYCLE	Deterministic	CO-PATH	Deterministic
2010	Chen et al. [4]	$O^*(3.24^k)$	Yes	$O^*(3.24^k)$	Yes
2015	Feng et al. [8]	$O^*(3^k)$	No	$O^*(3^k)$	No
2016	Xiao [22]	$O^*(3.07^k)$	Yes	–	–
2022	Tsur [19]	$O^*(3^k)$	Yes	$O^*(3^k)$	Yes
2024	This paper	$O^*(2.8192^k)$	Yes	$O^*(2.9241^k)$	No

refined to $O^*(3.07^k)$ for CO-PATH/CYCLE PACKING by Xiao [22]. Feng et al. [8] introduced a randomized $O^*(3^k)$ -time algorithm for the CO-PATH PACKING, which also works for CO-PATH/CYCLE PACKING. However, we do not know how to derandomize this algorithm. Recently, Tsur [19] provided $O^*(3^k)$ -time algorithms solving CO-PATH/CYCLE PACKING and CO-PATH PACKING deterministically. It seems that the bound 3^k is hard to break for the two problems. As shown in Tsur’s algorithms [19], many cases, including the case of handling all degree-4 vertices, lead to the same bottleneck. One of the main targets in this paper is to break those bottlenecks. Previous results and our results for CO-PATH/CYCLE PACKING and CO-PATH PACKING are summarized in Table 1.

Our Contributions. The main contributions of this paper are a deterministic algorithm for CO-PATH/CYCLE PACKING running in $O^*(2.8192^k)$ time and $O^*(2.5199^k)$ space and a randomized algorithm for CO-PATH PACKING running in $O^*(2.9241^k)$ time and space with failure probability $\leq 1/3$. To obtain this result, we need to combine path decomposition, dynamic programming, branch-and-search and some other techniques. In the previous $O^*(3^k)$ -time algorithms for both CO-PATH/CYCLE PACKING and CO-PATH PACKING, many cases, including dealing with degree-4 vertices and several different types of degree-3 vertices, lead to the same bottleneck. It seems very hard to avoid all the bottleneck cases by simply modifying the previous algorithms. The main idea of the algorithm in this paper is as follows. We first design some new reduction and branching rules to handle some good structures of the graph. After this, we prove that the remaining graph has a small pathwidth and then design an efficient dynamic programming algorithm based on a path decomposition. Specifically, our algorithm firstly runs the branch-and-search algorithm to handle the degree- ≥ 5 vertices and the degree-4 vertices adjacent to at least one degree- ≥ 3 vertex. In the branch-and-search phase, our algorithm runs in $O^*(2.8192^k)$ time for both CO-PATH/CYCLE PACKING and CO-PATH PACKING. When branching steps cannot be applied, we can construct a path decomposition of width at most $2k/3 + \epsilon k$ for any $\epsilon > 0$ and call our dynamic programming algorithm. The running time and space of the dynamic programming algorithm are bounded by $O^*(2.5199^k)$ for CO-PATH/CYCLE PACKING and bounded by $O^*(2.9241^k)$ with failure probability $\leq 1/3$ for CO-PATH PACKING. Therefore, the whole algorithm runs in $O^*(2.8192^k)$ time and $O^*(2.5199^k)$ space for CO-PATH/CYCLE PACKING and runs in $O^*(2.9241^k)$ time and space with failure probability $\leq 1/3$ for CO-PATH PACKING.

For the dynamic programming algorithms based on a path decomposition, the first algorithm is an $O^*((d+2)^p)$ -time algorithm to solve d -BOUNDED-DEGREE VERTEX DELETION for every $d \geq 1$, where p is the width of the given path decomposition. This was firstly found in [21]. We also present it in our way to make this paper self-contained. The second algorithm is designed for CO-PATH PACKING. In this algorithm, we use an algorithm framework called cut & count [6]. Given a path decomposition of width p , we show that CO-PATH PACKING can be solved in $O^*(5^p)$ time and space with failure probability $\leq 1/3$.

Reading Guide. Section 2 begins with a review of fundamental definitions and the establishment of notation. In Section 3 we solve CO-PATH/CYCLE PACKING and in section 4 we solve CO-PATH PACKING. In Section 3.1, we show that a proper graph has a small pathwidth and present a dynamic programming algorithm for CO-PATH/CYCLE PACKING. In Section 3.2, we give the branch-and-search algorithm for CO-PATH/CYCLE PACKING. Similarly, in Section 4.1, we present a randomized dynamic programming algorithm for CO-PATH PACKING. In Section 4.2, we give the branch-and-search algorithm for CO-PATH PACKING. Most of the content of this branching algorithm is the same as the branching algorithm for CO-PATH/CYCLE PACKING. In Section 5, we give a conclusion. Due to lack of space, the proof of theorems marked ♣ is omitted here and they can be found in the full version of this paper [15].

2 Preliminaries

In this paper, we only consider simple and undirected graphs. Let $G = (V, E)$ be a graph with $n = |V|$ vertices and $m = |E|$ edges. A vertex v is called a *neighbor* of a vertex u if there is an edge $uv \in E$. Let $N(v)$ denote the set of neighbors of v . For a subset of vertices X , let $N(X) = \bigcup_{v \in X} N(v) \setminus X$ and $N[X] = N(X) \cup X$. We use $d(v) = |N(v)|$ to denote the *degree* of a vertex v in G . A vertex of degree d is called a *degree- d vertex*. For a subset of vertices $X \subseteq V$, the subgraph induced by X is denoted by $G[X]$. The induced subgraph $G[V \setminus X]$ is also written as $G \setminus X$. A *path* P in G is a sequence of vertices v_1, v_2, \dots, v_t such that for any $1 \leq i < t$, $\{v_i v_{i+1}\} \in E$. Two vertices u and v are *reachable* to each other if there is a path v_1, v_2, \dots, v_t such that $v_1 = u$ and $v_t = v$. A *connected component* of a graph is a maximum subgraph such that any two vertices are reachable to each other. A vertex subset S is called a *cPCP-set* of graph G if the degree of any vertex in $G \setminus S$ is at most 2. A vertex subset S is called a *cPP-set* of graph G if $G \setminus S$ is a collection of disjoint paths. For a graph G , we will use $V(G)$ and $E(G)$ to denote the vertex set and edge set of it, respectively. A complete graph with 3 vertices is called a *triangle*. A singleton $\{v\}$ may be denoted as v .

Path decomposition. We will use the concepts of path decomposition and nice path decomposition of a graph.

► **Definition 1** ([5]). *A path decomposition of a graph G is a sequence $P = (X_1, X_2, \dots, X_r)$ of vertex subsets $X_i \subseteq V(G)$ ($i \in \{1, 2, \dots, r\}$) such that:*

(P1) $\bigcup_{i=1}^r X_i = V(G)$.

(P2) *For every $uv \in E(G)$, there exists $l \in \{1, 2, \dots, r\}$ such that X_l contains both u and v .*

(P3) *For every $u \in V(G)$, if $u \in X_i \cap X_k$ for some $i \leq k$, then $u \in X_j$ for all $i \leq j \leq k$.*

For a path decomposition (X_1, X_2, \dots, X_r) of a graph, each vertex subset X_i in it is called a *bag*. The *width* of the path decomposition is $\max_i \{|X_i|\} - 1$. The *pathwidth* of a graph G , denoted by $\text{pw}(G)$, is the minimum possible width of a path decomposition of G . A path decomposition (X_1, X_2, \dots, X_r) is *nice* if $X_1 = X_r = \emptyset$ and for every $i \in \{1, 2, \dots, r-1\}$ there is either a vertex $v \notin X_i$ such that $X_{i+1} = X_i \cup \{v\}$, or there is a vertex $w \in X_i$ such that $X_{i+1} = X_i \setminus \{w\}$. The following lemma shows that any path decomposition can be turned into a nice path decomposition without increasing the width.

► **Lemma 2** ([5]). *If a graph G admits a path decomposition of width at most p , then it also admits a nice path decomposition of width at most p . Moreover, given a path decomposition $P = (X_1, X_2, \dots, X_r)$ of G of width at most p , one can in time $O(p^2 \cdot \max(r, |V(G)|))$ compute a nice path decomposition of G of width at most p .*

There are also easy ways to reduce the length r of a path decomposition to a polynomial of the graph size. Next, we will also assume that r is bounded by a polynomial of the number of vertices. In terms of the pathwidth, there is a known bound.

► **Theorem 3** ([10]). *For any $\epsilon > 0$, there exists an integer n_ϵ such that for every graph G with $n > n_\epsilon$ vertices,*

$$pw(G) \leq \frac{1}{6}n_3 + \frac{1}{3}n_4 + n_{\geq 5} + \epsilon n,$$

where n_i $i \in \{3, 4\}$ is the number of vertices of degree i in G and $n_{\geq 5}$ is the number of vertices of degree at least 5. Moreover, a path decomposition of the corresponding width can be constructed in polynomial time.

Branch-and-Search Algorithm. For a branch-and-search algorithm, we use a parameter k of the instance to measure the running time of the algorithm. Let $T(k)$ denote the maximum size of the search tree generated by the algorithm when running on an instance with the parameter no greater than k . Assume that a branching operation generates l branches and the measure k in the i -th instance decreases by at least c_i . This operation generates a recurrence relation

$$T(k) \leq T(k - c_1) + T(k - c_2) + \dots + T(k - c_l) + 1.$$

The largest root of the function $f(x) = 1 - \sum_{i=1}^l x^{-c_i}$ is called the *branching factor* of the recurrence. Let γ denote the maximum branching factor among all branching factors in the search tree. The running time of the algorithm is bounded by $O^*(\gamma^k)$. For more details about analyzing branch-and-search algorithms, please refer to [13].

3 A Parameterized Algorithm for Co-Path/Cycle Packing

In this section, we propose a parameterized algorithm for CO-PATH/CYCLE PACKING. First, in Section 3.1, we show that CO-PATH/CYCLE PACKING on a special graph class, called *proper graph*, can be quickly solved by using the dynamic programming algorithm based on path decompositions in Theorem 6. The key point in this section is to bound the pathwidth of proper graphs by $2k/3 + \epsilon k$ for any $\epsilon > 0$. Second, in Section 3.2, we give a branch-and-search algorithm that will implement some branching steps on special local graph structures. When there is no good structure to apply our branching rules, we show that the graph must be a proper graph and then the algorithm in Section 3.1 can be called directly to solve the problem.

3.1 Proper Graphs with Small Pathwidth

A graph is called *proper* if it satisfies the following conditions:

1. The maximum degree of G is at most 4.
2. For any degree-4 vertex v , all neighbors are of degree at most 2.
3. For any degree-2 vertex v , at least one vertex in $N(v)$ is of degree at least 3.
4. Each connected component contains at least 6 vertices.

We are going to solve CO-PATH/CYCLE PACKING on proper graphs first. Next, we try to bound the pathwidth of proper graphs.

11:6 Solving Co-Path/Cycle Packing and Co-Path Packing Faster Than 3^k

► **Lemma 4** (♣). *Let G be a proper graph. If G has a cPCP-set (resp. cPP-set) of size at most k , then it holds that*

$$|V(G)| \leq 100k \quad \text{and} \quad \frac{n_3}{6} + \frac{n_4}{3} \leq \frac{2k}{3}, \quad (1)$$

where n_3 and n_4 are the number of degree-3 and degree-4 vertices in G , respectively.

► **Lemma 5** (♣). *Let G be a proper graph. For any $\epsilon > 0$, in polynomial time we can either decide that G has no cPCP-set (resp. cPP-set) of size at most k or compute a path decomposition of width at most $\frac{2k}{3} + \epsilon k$.*

The following theorem shows that there exists an algorithm for the general d -BOUNDED-DEGREE VERTEX DELETION problem based on a given path decomposition of the graph. The running time bound of the algorithm is $O^*((d+2)^p)$, where p is the width of the given path decomposition. Previously, an $O^*(3^p)$ -time algorithm for 1-BOUNDED-DEGREE VERTEX DELETION was known [2]. Recently, this result was extended for any $d \geq 1$ by van Rooij [21]. We also present it (in the full version of this paper) in our way to make this paper self-contained.

► **Theorem 6** (♣). *Given a path decomposition of G with width p . For any $d \geq 1$, d -BOUNDED-DEGREE VERTEX DELETION can be solved in $O^*((d+2)^p)$ time and space.*

However, the algorithm given in Theorem 6 cannot be used to solve CO-PATH PACKING since there is a global connectivity constraint for CO-PATH PACKING. We will discuss it in Section 4. Based on Theorem 6, we have the following lemma.

► **Lemma 7**. *CO-PATH/CYCLE PACKING on proper graphs can be solved in $O^*(2.5199^k)$ time.*

Proof. We first call the algorithm in Lemma 5. If the algorithm decides that G has no cPCP-set of size at most k , we claim that (G, k) is a no-instance. Otherwise, we can obtain a nice path decomposition of width at most $\frac{2k}{3} + \epsilon k$. Then we call the algorithm in Theorem 6. This algorithm runs in $O^*(4^{2k/3 + \epsilon k}) = O^*(2.5199^k)$, where we choose $\epsilon < 10^{-6}$. This lemma holds. ◀

3.2 A Branch-and-Search Algorithm

In this subsection, we provide a branch-and-search algorithm for CO-PATH/CYCLE PACKING, which is denoted by $\text{cPCP}(G, k)$. Our algorithm contains several reduction and branching steps. After recursively executing these steps, we will get a proper graph and then call the dynamic programming algorithm in Section 3.1 to solve it.

3.2.1 Reduction and Branching Rules

Firstly we have a reduction rule to reduce small connected components.

► **Reduction-Rule 1.** *If there is a connected component C of the graph such that $|V(C)| \leq 6$, then run a brute force algorithm to find a minimum cPCP-set S in C , delete C and include S in the deletion set.*

► **Lemma 8.** *Let u and v be two adjacent vertices of degree at most 2 in G and G' be the graph after deleting edge uv from G . Then (G, k) is a yes-instance if and only if (G', k) is a yes-instance.*

This lemma holds because adding an edge between two vertices of degree at most 1 back to a graph will not make the two vertices of degree greater than 2. Based on this lemma, we have the following reduction rule.

► **Reduction-Rule 2.** *If there are two adjacent vertices u and v of degree at most 2, then return $\text{cPCP}(G' = (V(G), E(G) \setminus \{uv\}), k)$.*

► **Lemma 9.** *Let $\{u, v, w\}$ be a triangle such that $|N(\{u, v, w\})| = 1$. Let x be the vertex in $N(\{u, v, w\})$. There is a minimum cPCP-set containing x .*

This lemma holds because we must delete at least one vertex in $\{u, v, w, x\}$ and delete any vertex in $\{u, v, w\}$ cannot decrease the degree of vertices in $V(G) \setminus \{u, v, w, x\}$. Based on this lemma, we have the following reduction rule.

► **Reduction-Rule 3.** *If there is a triangle $\{u, v, w\}$ such that $|N(\{u, v, w\})| = 1$, then return $\text{cPCP}(G \setminus N[\{u, v, w\}], k - 1)$.*

After applying the three simple reduction rules, we will execute some branching steps. Although we have several branching steps, most of them are based on the following two branching rules.

For a vertex v of degree at least 3, either it is included in the deletion set or it remains in the graph. For the latter case, there are at most two vertices in $N(v)$ that can remain in the graph. So we have the following branching rule.

Branching-Rule (B1). *For a vertex v of degree at least 3, branch on it to generate $\binom{|N(v)|}{2} + 1$ branches by either (i) deleting v from the graph and including it in the deletion set, or (ii) for every pair of vertices u and w in $N(v)$, deleting $N(v) \setminus \{u, w\}$ from the graph and including $N(v) \setminus \{u, w\}$ in the deletion set.*

A vertex v dominates a vertex u if $N[u] \subseteq N[v]$. We have the following property for dominated vertices.

► **Lemma 10.** *If a vertex v dominates a vertex u , then there is a minimum cPCP-set either containing v or containing none of v and u .*

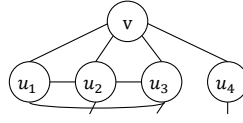
Proof. Let S be a cPCP-set. Assume to the contrary that $S \cap \{v, u\} = \{u\}$. Since v dominates u , we know that $S' = S \setminus \{u\} \cup \{v\}$ is still a feasible cPCP-set with $|S'| = |S|$. There is a minimum cPCP-set containing v . This lemma holds. ◀

Assume that v dominates u . By Lemma 10, we know that either v is included in the deletion set or both of v and u remain in the graph. For the latter case, there are at most two vertices in $N(v)$ that can remain in the graph. Thus, at least $|N(v)| - 2$ vertices in $N(v) \setminus \{u\}$ will be deleted. We get the following branching rule.

Branching-Rule (B2). *Assume that a vertex v of degree at least 3 dominates a vertex u . Branch on v to generate $1 + (|N(v)| - 1) = |N(v)|$ branches by either (i) deleting v from the graph and including it in the deletion set, or (ii) for each vertex $w \in N(v) \setminus \{u\}$, deleting $N(v) \setminus \{u, w\}$ from the graph and including $N(v) \setminus \{u, w\}$ in the deletion set.*

3.2.2 Steps

When we execute one step, we assume that all previous steps are not applicable in the current graph anymore. We will analyze each step after describing it.



■ **Figure 1** Degree-4 vertex v dominates a degree-3 vertex u_1 .

Step 1 (Vertices of degree at least 5). If there is a vertex v of $d(v) \geq 5$, then branch on v with Branching-Rule (B1) to generate $\binom{d(v)}{d(v)-2} + 1$ branches and return the best of

$$\text{cPCP}(G \setminus \{v\}, k - 1)$$

and $\text{cPCP}(G \setminus (N(v) \setminus \{u, w\}), k - |N(v) \setminus \{u, w\}|)$ for each pair $\{u, w\} \subseteq N(v)$.

For this step, we get a recurrence

$$T(k) \leq T(k - 1) + \binom{d(v)}{d(v) - 2} \times T(k - (d(v) - 2)) + 1,$$

where $d(v) \geq 5$. For the worst case that $d(v) = 5$, the branching factor of it is 2.5445.

After Step 1, the graph contains only vertices with degree at most 4.

Step 2 (Degree-4 vertices dominating some vertex of degree at least 3). Assume that there is a degree-4 vertex v that dominates a vertex u_1 , where $d(u_1) \geq 3$. Without loss of generality, we assume that the other three neighbors of v are u_2, u_3 and u_4 , and u_1 is adjacent to u_2 and u_3 (u_1 is further adjacent to u_4 if $d(u_1) = 4$). See Figure 1 for an illustration. We branch on v with Branching-Rule (B2) and get $|N(v)|$ branches

$$\text{cPCP}(G \setminus \{v\}, k - 1), \text{cPCP}(G \setminus \{u_2, u_3\}, k - 2),$$

$$\text{cPCP}(G \setminus \{u_2, u_4\}, k - 2), \text{ and } \text{cPCP}(G \setminus \{u_3, u_4\}, k - 2).$$

The corresponding recurrence is

$$T(k) \leq T(k - 1) + 3 \times T(k - 2) + 1,$$

which has a branching factor of 2.3028.

A triangle $\{u, v, w\}$ is called *heavy* if it holds that $|N(\{u, v, w\})| \geq 4$.

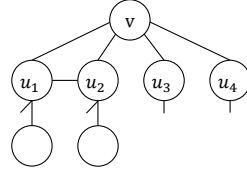
Step 3 (Degree-4 vertices in a heavy triangle). Assume that a degree-4 vertex v is in a heavy triangle. Let u_1, u_2, u_3 and u_4 be the four neighbors of v , where we assume without loss of generality that $\{v, u_1, u_2\}$ is a heavy triangle. See Figure 2 for an illustration. We branch on v with Branching-Rule (B1). In the branch of deleting $\{u_3, u_4\}$, we can simply assume that the three vertices v, u_1 and u_2 are not deleted, otherwise this branch can be covered by another branch and then it can be ignored. Since v, u_1 and u_2 form a triangle, we need to delete all the vertices in $N(\{v, u_1, u_2\})$ in this branch. Note that $\{v, u_1, u_2\}$ is a heavy triangle and we have that $|N(\{v, u_1, u_2\})| \geq 4$. We generate the following $\binom{4}{2} + 1$ branches

$$\text{cPCP}(G \setminus \{v\}, k - 1),$$

$$\text{cPCP}(G \setminus (\{u_1, u_i\}), k - |\{u_1, u_i\}|) \text{ for each } i = 2, 3, 4,$$

$$\text{cPCP}(G \setminus (\{u_2, u_i\}), k - |\{u_2, u_i\}|) \text{ for each } i = 3, 4,$$

and $\text{cPCP}(G \setminus N(\{v, u_1, u_2\}), k - |N(\{v, u_1, u_2\})|)$.



■ **Figure 2** Degree-4 vertex v is in a heavy triangle $\{v, u_1, u_2\}$.

The corresponding recurrence is

$$T(k) \leq T(k-1) + \binom{4}{2} - 1 \times T(k-2) + T(k - |N(\{v, u_1, u_2\})|) + 1,$$

where $|N(\{v, u_1, u_2\})| \geq 4$. For the worst case that $|N(\{v, u_1, u_2\})| = 4$, the branching factor of it is 2.8186.

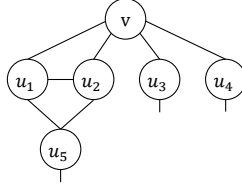
► **Lemma 11.** *If a vertex v has two degree-1 neighbors u_1 and u_2 , then there is a minimum cPCP-set containing none of u_1 and u_2 . Furthermore, If v is a vertex of degree at most 3, then there is a minimum cPCP-set containing none of v , u_1 , and u_2 .*

Proof. Let S be a minimum cPCP-set. If $v \in S$, then $S \setminus \{u_1, u_2\}$ is still a minimum cPCP-set. Next, we assume that $v \notin S$. If at least one of u_1 and u_2 is in S , then $S' = (S \setminus \{u_1, u_2\}) \cup \{v\}$ is a cPCP-set with $|S'| \leq |S|$. Thus, S' is a minimum cPCP-set not containing u_1 and u_2 .

If v is a degree-2 vertex, then the component containing v is a path of three vertices. None of the three vertices should be deleted. If v is a degree-3 vertex, we let u_3 be the third neighbor of v . Note that at least one vertex in $\{v, u_1, u_2, u_3\}$ should be deleted and then any solution S will contain at least one vertex in $\{v, u_1, u_2, u_3\}$. We can see that $S' = (S \setminus \{v, u_1, u_2, u_3\}) \cup \{u_3\}$ is still a cPCP-set with size $|S'| \leq |S|$. There is always a minimum cPCP-set containing none of v , u_1 , and u_2 . ◀

Step 4 (Degree-4 vertices in a triangle). Assume that there is still a degree-4 vertex v in a triangle $\{v, u_1, u_2\}$. We also let u_1, u_2, u_3 and u_4 be the four neighbors of v . Since triangle $\{v, u_1, u_2\}$ can not be a heavy triangle now, we know that $|N(\{v, u_1, u_2\})| \leq 3$. First, we show that it is impossible $|N(\{v, u_1, u_2\})| = 2$. Assume to the contrary that $|N(\{v, u_1, u_2\})| = 2$. Then u_1 and u_2 can only be adjacent to vertices in $N[v]$. If both of u_1 and u_2 are degree-2 vertices, then Reduction-Rule 1 should be applied. If one of u_1 and u_2 is a vertex of degree at least 3, then v would dominate this vertex, and then the condition of Step 2 would hold. Any case is impossible. Next, we assume that $|N(\{v, u_1, u_2\})| = 3$ and let u_3 denote the third vertex in $N(\{v, u_1, u_2\})$. We further consider several different cases.

Case 1: One of u_1 and u_2 , say u_1 is a degree-4 vertex. For this case, vertex u_1 is adjacent to u_5 , otherwise the degree-4 vertex v would dominate the degree-4 vertex u_1 . Since u_1 is of degree 4, we know that u_1 is also adjacent to one of u_3 and u_4 , say u_3 . Vertices u_2 and u_3 can only be adjacent to vertices in $N[v] \cup \{u_5\}$, otherwise $\{v, u_1, u_2\}$ or $\{v, u_1, u_3\}$ would form a heavy triangle and Step 3 should be applied. Thus, neither u_2 nor u_3 can be a degree-3 vertex, otherwise degree-4 vertex u_1 or v dominates a degree-3 vertex u_2 or u_3 and then Step 2 should be applied. Thus, we have that either $d(u_2) = d(u_3) = 2$ or $d(u_2) = d(u_3) = 4$. We further consider the following three cases:



■ **Figure 3** In the Case 2 of Step 4, degree-4 vertex v is in a triangle $\{v, u_1, u_2\}$ and both of u_1 and u_2 are degree-3 vertices.

Case 1.1: $d(u_2) = d(u_3) = 2$. For this case, we have that v dominates u_2 and u_3 . By Lemma 10, we have that there is a minimum cPCP-set either containing v or containing none of v , u_2 and u_3 . For the first branch, we delete v from the graph and include it in the deletion set. For the second branch, we delete u_1 and u_4 from the graph and include it in the deletion set. The corresponding recurrence is

$$T(k) \leq T(k-1) + T(k-2) + 1,$$

the branching factor of which is 1.6181.

Case 1.2: $d(u_2) = d(u_3) = 4$ and u_2 and u_3 are not adjacent. For this case, both of u_2 and u_3 are adjacent to u_4 and u_5 . Since v does not dominate u_4 , we know that u_4 is adjacent to at least one vertex out of $N[v]$. Since triangle $\{v, u_2, u_4\}$ is not a heavy triangle, we know that u_4 is not adjacent to any vertex other than $N[v] \cup \{u_5\}$. Thus, u_4 is also adjacent to u_5 . Since the maximum degree of the graph is 4 now, we know that this component only contains six degree-4 vertices $N[v] \cup \{u_5\}$, which should be eliminated by Reduction-Rule 2.

Case 1.3: $d(u_2) = d(u_3) = 4$ and u_2 and u_3 are adjacent. For this case, since v does not dominate u_3 , we know that u_3 is adjacent to at least one vertex out of $N[v]$. This vertex can only be u_5 . Thus, the four neighbors of u_3 are v, u_1, u_2 , and u_5 . Now u_1 is a degree-4 vertex dominating a degree-4 vertex u_3 . Step 2 should be applied. Thus, this case is impossible.

Case 2: Both of u_1 and u_2 are degree-3 vertices. It holds that $N(u_1) = \{v, u_2, u_5\}$ and $N(u_2) = \{v, u_1, u_5\}$, otherwise vertex v would dominate u_1 or u_2 . See Figure 3 for an illustration. For this case, we first branch on v with Branching-Rule (B1) to generate $\binom{d(v)}{d(v)-2} + 1$ branches. We can only get a recurrence relation

$$T(k) \leq T(k-1) + \binom{4}{2} T(k-2) + 1.$$

This recurrence is not good enough. Next, we look at the branch of deleting v and try to get some improvements on this subbranch. After deleting v , vertices u_1 and u_2 become degree-2 vertices in a triangle. We first apply Reduction-Rule 1 to delete edge $u_1 u_2$ between u_1 and u_2 . Then, vertices u_1 and u_2 become degree-1 vertices adjacent to u_5 .

Case 2.1: u_5 is a degree-2 vertex. This case is impossible otherwise Reduction Rule 3 would be applied before this step.

Case 2.2: u_5 is a degree-3 vertex. By Lemma 11, we can delete $N[u_5]$ directly and include the third neighbor of u_5 in the deletion set. We generate the following $\binom{4}{2} + 1$ branches

$$\begin{aligned} & \text{cPCP}(G \setminus (\{v\} \cup N[u_5]), k-2) \\ & \text{and } \text{cPCP}(G \setminus (N(v) \setminus \{u, w\}), k - |N(v) \setminus \{u, w\}|) \text{ for each pair } \{u, w\} \subseteq N(v). \end{aligned}$$

The corresponding recurrence is

$$T(k) \leq T(k-2) + \binom{4}{2}T(k-2) + 1.$$

which has a branching factor of 2.6458.

Case 2.3: u_5 is a degree-4 vertex. By Lemma 11, we know that there is a minimum cPCP-set either containing u_5 or containing none of u_5, u_1 , and u_2 . For the first case, we delete u_5 and include it in the deletion set. For the second case, we delete $N[u_5]$ from the graph and include $N(u_5) \setminus \{u_5, u_1, u_2\}$ in the deletion set. Note that $|N(u_5) \setminus \{u_5, u_1, u_2\}| = 2$ since u_5 is a degree-4 vertex. Combining with the previous branching on v , we get the following $\binom{4}{2} + 2$ branches

$$\begin{aligned} & \text{cPCP}(G \setminus \{v, u_5\}, k-2), \\ & \text{cPCP}(G \setminus (\{v\} \cup N[u_5]), k-3), \end{aligned}$$

and $\text{cPCP}(G \setminus (N(v) \setminus \{u, w\}), k - |N(v) \setminus \{u, w\}|)$ for each pair $\{u, w\} \subseteq N(v)$.

The corresponding recurrence is

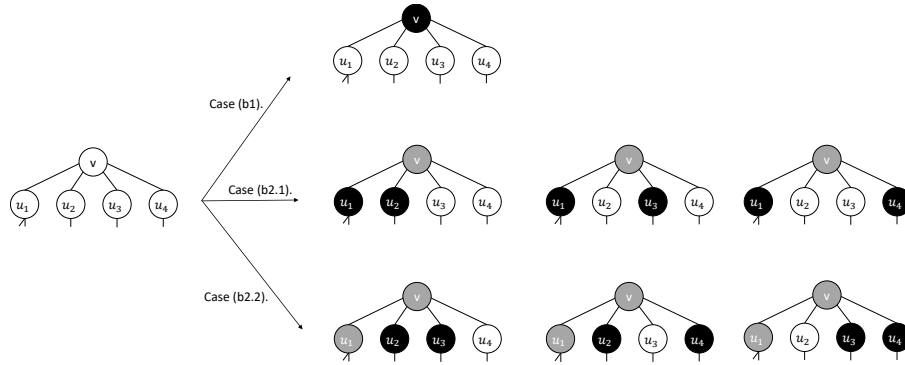
$$T(k) \leq T(k-1-1) + T(k-1-2) + \binom{4}{2}T(k-2) + 1,$$

which has a branching factor of 2.7145.

After Step 4, no degree-4 vertex is in a triangle.

Step 5 (Degree-4 vertices adjacent to some vertex of degree at least 3). Assume there is a degree-4 vertex v adjacent to at least one vertex of degree at least 3. Let u_1, u_2, u_3 and u_4 be the four neighbors of v , where we assume without loss of generality that $d(u_1) \geq 3$. First, we branch on v by either (b1) including it in the solution set or (b2) excluding it from the solution set. Next, we focus on the latter case (b2). In (b2), we further branch on u_1 by (b2.1) either including it in the solution set or (b2.2) excluding it from the solution set. For case (b2.1), there are at least $|N(v)| - 2 = 2$ vertices in $N(v)$ that should be deleted by the same argument for Branching-Rule (B1). Thus, we can generate three subbranches by deleting $\{u_1, u_2\}$, $\{u_1, u_3\}$, and $\{u_1, u_4\}$, respectively. For case (b2.2), there are still at least $|N(v)| - 2 = 2$ vertices in $N(v)$ that should be deleted, which can be one of the following three sets $\{u_2, u_3\}$, $\{u_2, u_4\}$, and $\{u_3, u_4\}$. Furthermore, there are also at least $|N(u_1)| - 2 \geq 1$ vertices in $N(u_1)$ that should be deleted. Note that $v \in N(u_1)$ is not allowed to be deleted now. Thus, at least $|N(u_1) \setminus \{v\}| - 1$ vertices in $N(u_1) \setminus \{v\}$ should be deleted. We will generate $\binom{|N(u_1) \setminus \{v\}|}{|N(u_1) \setminus \{v\}| - 1} = |N(u_1) \setminus \{v\}| = d(u_1) - 1$ branches by decreasing k by $|N(u_1) \setminus \{v\}| - 1 = d(u_1) - 2$. Since after Step 3, the degree-4 vertex v is not in any triangle, we know that $N(u_1) \setminus \{v\}$ is disjoint with $\{u_2, u_3, u_4\}$. For case (b2.2), we will generate $3 \times (d(u_1) - 1)$ subbranches by decreasing k by at least $2 + d(u_1) - 2 = d(u_1)$ in each, where $d(u_1) = 3$ or 4 . See Figure 4 for an illustration. In total, we will generate the following $1 + 3 + 3 \times (d(u_1) - 1)$ branches

$$\begin{aligned} & \text{cPCP}(G \setminus \{v\}, k-1), \\ & \text{cPCP}(G \setminus (\{u_1, u_i\}), k-2) \text{ for each } i = 2, 3, 4, \\ & \text{cPCP}(G \setminus (\{u_2, u_3\} \cup (N(u_1) \setminus \{v, w\})), k - d(u_1)), \text{ for each } w \in N(u_1) \setminus \{v\}, \\ & \text{cPCP}(G \setminus (\{u_2, u_4\} \cup (N(u_1) \setminus \{v, w\})), k - d(u_1)), \text{ for each } w \in N(u_1) \setminus \{v\}, \\ & \text{and } \text{cPCP}(G \setminus (\{u_3, u_4\} \cup (N(u_1) \setminus \{v, w\})), k - d(u_1)), \text{ for each } w \in N(u_1) \setminus \{v\}. \end{aligned}$$



■ **Figure 4** Vertices in the deletion set are denoted by black vertices, and vertices not allowed to be deleted are denoted by grey vertices.

■ **Table 2** The branching factors of each of the first five steps.

Steps	Step 1	Step 2	Step 3	Step 4	Step 5
Branching factors	2.5445	2.3028	2.8186	2.7145	2.8192

We get a recurrence

$$T(k) \leq T(k - 1) + 3 \times T(k - 2) + 3(d(u_1) - 1) \times T(k - d(u_1)) + 1,$$

where $d(u_1) = 3$ or 4 . For the case that $d(u_1) = 3$, the branching factor is 2.8192 , and for the case that $d(u_1) = 4$, the branching factor is 2.6328 .

The worst branching factors in the above five steps are listed in Table 2. After Step 5, any degree-4 vertex can be only adjacent to vertices of degree at most 2. It is easy to see that the remaining graph after Step 5 is a proper graph. We call the algorithm in Lemma 7 to solve the instance in $O^*(2.5199^k)$ time.

► **Theorem 12.** *CO-PATH/CYCLE PACKING can be solved in $O^*(2.8192^k)$ time.*

Fomin et al. [9] introduced the monotone local search technique for deriving exact exponential-time algorithms from parameterized algorithms. Under some conditions, a $c^k n^{O(1)}$ -time algorithm implies an algorithm with running time $(2 - 1/c)^{n+o(n)}$. By applying this technique on our $O^*(2.8192^k)$ -time algorithm in Theorem 12, we know that CO-PATH/CYCLE PACKING can be solved in $(2 - 1/2.8192)^{n+o(n)} = O(1.6453^n)$ time.

► **Corollary 13.** *CO-PATH/CYCLE PACKING can be solved in $O(1.6453^n)$ time.*

4 A Parameterized Algorithm for Co-Path Packing

In this section, we show a randomized $O^*(2.9241^k)$ time algorithm for CO-PATH PACKING. We also use the method for CO-PATH/CYCLE PACKING, which consists of two phases. The first one is the dynamic programming phase, and the second one is the branch-and-search phase.

In the dynamic programming phase, it is more difficult to solve CO-PATH PACKING based on a path decomposition in comparison to CO-PATH/CYCLE PACKING. The reason is that CO-PATH/CYCLE PACKING involves only local constraints, which means that the object's properties can be verified by checking each vertex's neighborhood independently. For this problem, a typical dynamic programming approach can be used to design a $c^{pw(G)} |V(G)|^{O(1)}$

time algorithm straightforwardly. In contrast, there is a global connectivity constraint for CO-PATH PACKING. The problem with global connectivity constraint is also called *connectivity-type problem*. For connectivity-type problems, the typical dynamic programming approach has to keep track of all possible ways the solution can traverse the corresponding separator of the path decomposition that is $\Omega(l^l)$, where l is the size of the separator and hence the pathwidth [6].

To obtain a single exponential algorithm parametrized by pathwidth for CO-PATH PACKING, we use the cut & count framework. Previously, cut & count significantly improved the known bounds for various well-studied connectivity-type problems, such as HAMILTONIAN PATH, STEINER TREE, and FEEDBACK VERTEX SET [6]. Additionally, for k -CO-PATH SET, cut & count has been used to obtain a fast parameterized algorithm [17]. In Section 4.1, we present a randomized fpt algorithm with complexity $O^*(5^{pw(G)})$ for CO-PATH PACKING.

In the branch-and-search phase, our algorithm for CO-PATH PACKING is similar to the branching algorithm for CO-PATH/CYCLE PACKING. Specifically, our branching algorithm for CO-PATH PACKING contains two reduction rules, two branching rules, and four steps, while the first reduction rule, both two branching rules, the first two steps, and the last step are the same as the branching algorithm for CO-PATH/CYCLE PACKING. The branch-and-search phase is shown in Section 4.2.

4.1 A DP Algorithm via Cut & Count for Co-Path Packing

In this section, we use the cut & count framework to design a $5^{pw}n^{O(1)}$ one-sided error Monte Carlo algorithm with a constant probability of a false negative for CO-PATH PACKING. Due to the limited space, we do not provide the proof of the following theorem here.

► **Theorem 14 (♣).** *Given a path decomposition of G with width p . CO-PATH PACKING can be solved in $O^*(5^p)$ time and space with failure probability $\leq 1/3$.*

4.2 The Whole Algorithm

In this section, we propose a parameterized algorithm for CO-PATH PACKING. First, in Section 4.2.1, we show that CO-PATH PACKING on a proper graph class can be quickly solved by using the dynamic programming algorithm based on path decompositions in Theorem 14. Similarly, we will use Lemma 5 to conclude this result. Second, in Section 4.2.2, we give a branch-and-search algorithm that will implement some branching steps on special local graph structures. Our branching algorithm for CO-PATH PACKING is similar to the branching algorithm for CO-PATH/CYCLE PACKING. Specifically, our branching algorithm for CO-PATH PACKING contains two reduction rules, two branching rules, and four steps, while the first reduction rule, both two branching rules, the first two steps and the last step are the same as the branching algorithm for CO-PATH/CYCLE PACKING. When all the steps cannot be applied, we show that the graph must be a proper graph and then the algorithm in Section 4.1 can be called directly to solve the problems.

4.2.1 Proper Graphs with Small Pathwidth

Recall that a graph is called proper if it satisfies the following conditions:

1. The maximum degree of G is at most 4.
2. For any degree-4 vertex v , all neighbors are of degree at most 2.
3. For any degree-2 vertex v , at least one vertex in $N(v)$ is of degree at least 3.
4. Each connected component contains at least 6 vertices.

Combine Lemma 5 for CO-PATH PACKING with Lemma 14, we have the following lemma.

► **Lemma 15 (♣).** *CO-PATH PACKING on proper graphs can be solved in $O^*(2.9241^k)$ time with probability at least $2/3$.*

4.2.2 A Branch-and-Search Algorithm

In this subsection, we provide a branch-and-search algorithm for CO-PATH PACKING, which is denoted by $\text{cPP}(G, k)$. Our algorithm contains several reduction and branching steps. After recursively executing these steps, we will get a proper graph and then call the dynamic programming algorithm in Lemma 15 to solve it.

4.2.2.1. Reduction and Branching Rules. Firstly, we present two reduction rules. The first reduction rule is Reduction Rule 1 for CO-PATH/CYCLE PACKING and Reduction Rule *2 is a new reduction rule for CO-PATH PACKING.

Reduction-Rule 1. *If there is a connected component C of the graph such that $|V(C)| \leq 6$, then run a brute force algorithm to find a minimum cPP-set S in C , delete C and include S in the deletion set.*

A path $v_0v_1 \dots v_{h-1}v_h$ is called a *degree-two-path* if the two vertices v_0 and v_h are of degree not 2 and the other vertices $v_1 \dots v_{h-1}$ are of degree 2, where we allow $v_0 = v_h$.

► **Lemma 16 (♣).** *For a degree-two-path $P = v_0v_1 \dots v_{h-1}v_h$ with $h \geq 4$ in $G = (V, E)$, let $G' = (V \setminus v_2, E \setminus \{v_1v_2, v_2v_3\} \cup \{v_1v_3\})$, we have that (G, k) is a yes-instance if and only if (G', k) is a yes-instance.*

Based on this lemma, we have the following reduction rule for CO-PATH PACKING.

Reduction-Rule *2. *If there is a degree-two-path $P = v_0v_1 \dots v_{h-1}v_h$ with $h \geq 4$, then return $\text{cPP}(G' = (V \setminus v_2, E \setminus \{v_1v_2, v_2v_3\} \cup \{v_1v_3\}), k)$.*

If Reduction Rules 1 and *2 cannot be applied, we have a property that for any degree-2 vertex v , at least one vertex in $N(v)$ is of degree at least 3.

After applying the three simple reduction rules, we will execute some branching steps. For CO-PATH PACKING, we have two branching rules, which are the same as the two branching rules for CO-PATH/CYCLE PACKING. The correctnesses of these two branching rules for CO-PATH PACKING are similar to the two branching rules for CO-PATH/CYCLE PACKING.

Branching-Rule (B1). *For a vertex v of degree at least 3, branch on it to generate $\binom{|N(v)|}{2} + 1$ branches by either (i) deleting v from the graph and including it in the deletion set, or (ii) for every pair of vertices u and w in $N(v)$, deleting $N(v) \setminus \{u, w\}$ from the graph and including $N(v) \setminus \{u, w\}$ in the deletion set.*

Branching-Rule (B2). *Assume that a vertex v of degree at least 3 dominates a vertex u . Branch on v to generate $1 + (|N(v)| - 1) = |N(v)|$ branches by either (i) deleting v from the graph and including it in the deletion set, or (ii) for each vertex $w \in N(v) \setminus \{u\}$, deleting $N(v) \setminus \{u, w\}$ from the graph and including $N(v) \setminus \{u, w\}$ in the deletion set.*

4.2.2.2. Steps. When we execute one step, we assume that all previous steps are not applicable in the current graph anymore. In this subsection, we present four steps: Step 1, Step 2, Step *3 and Step *4 for CO-PATH PACKING. Steps 1 and 2 are the Steps 1 and 2 for CO-PATH/CYCLE PACKING. Step *3 is a new step designed for CO-PATH PACKING. Step *4 is the Step 5 for CO-PATH/CYCLE PACKING.

Step 1 (Vertices of degree at least 5). If there is a vertex v of $d(v) \geq 5$, then branch on v with Branching-Rule (B1) to generate $\binom{d(v)}{d(v)-2} + 1$ branches.

Step 2 (Degree-4 vertices dominating some vertex of degree at least 3). Assume that there is a degree-4 vertex v that dominates a vertex u_1 , where $d(u_1) \geq 3$. Without loss of generality, we assume that the other three neighbors of v are u_2, u_3 and u_4 , and u_1 is adjacent to u_2 and u_3 (u_1 is further adjacent to u_4 if $d(u_1) = 4$). We branch on v with Branching-Rule (B2) and get $|N(v)|$ branches.

Step *3 (Degree-4 vertices in a triangle). Assume that a degree-4 vertex v is in a triangle. Let u_1, u_2, u_3 and u_4 be the four neighbors of v , where we assume without loss of generality that $\{v, u_1, u_2\}$ is a triangle. We branch on v with Branching-Rule (B1). In the branch of deleting $\{u_3, u_4\}$, we can simply assume that the three vertices v, u_1 and u_2 are not deleted, otherwise this branch can be covered by another branch and then it can be ignored. Since v, u_1 and u_2 form a triangle, we know this case is impossible, so we can ignore this subbranch. We generate the following $\binom{4}{2}$ branches

$$\begin{aligned} & \text{cPP}(G \setminus \{v\}, k - 1), \\ & \text{cPP}(G \setminus (\{u_1, u_i\}), k - |\{u_1, u_i\}|) \text{ for each } i = 2, 3, 4, \\ & \text{and } \text{cPP}(G \setminus (\{u_2, u_i\}), k - |\{u_2, u_i\}|) \text{ for each } i = 3, 4. \end{aligned}$$

The corresponding recurrence is

$$T(k) \leq T(k - 1) + \left(\binom{4}{2} - 1 \right) \times T(k - 2) + 1.$$

The branching factor of it is 2.7913.

After Step *3, no degree-4 vertex is in a triangle.

Step *4 (Degree-4 vertices adjacent to some vertex of degree at least 3). Assume there is a degree-4 vertex v adjacent to at least one vertex of degree at least 3. Let u_1, u_2, u_3 and u_4 be the four neighbors of v , where we assume without loss of generality that $d(u_1) \geq 3$. First, we branch on v by either (b1) including it in the solution set or (b2) excluding it from the solution set. Next, we focus on the latter case (b2). In (b2), we further branch on u_1 by (b2.1) either including it in the solution set or (b2.2) excluding it from the solution set. For case (b2.1), there are at least $|N(v)| - 2 = 2$ vertices in $N(v)$ that should be deleted by the same argument for Branching-Rule (B1). Thus, we can generate three subbranches by deleting $\{u_1, u_2\}$, $\{u_1, u_3\}$, and $\{u_1, u_4\}$, respectively. For case (b2.2), there are still at least $|N(v)| - 2 = 2$ vertices in $N(v)$ that should be deleted, which can be one of the following three sets $\{u_2, u_3\}$, $\{u_2, u_4\}$, and $\{u_3, u_4\}$. Furthermore, there are also at least $|N(u_1)| - 2 \geq 1$ vertices in $N(u_1)$ that should be deleted. Note that $v \in N(u_1)$ is not allowed to be deleted now. Thus, at least $|N(u_1) \setminus \{v\}| - 1$ vertices in $N(u_1) \setminus \{v\}$ should be deleted. We will generate $\binom{|N(u_1) \setminus \{v\}|}{|N(u_1) \setminus \{v\}| - 1} = |N(u_1) \setminus \{v\}| = d(u_1) - 1$ branches by decreasing k by

■ **Table 3** The branching factors of each of the steps.

Steps	Step 1	Step 2	Step *3	Step *4
Branching factors	2.5445	2.3028	2.7913	2.8192

$|N(u_1) \setminus \{v\}| - 1 = d(u_1) - 2$. Since after Step *3, the degree-4 vertex v is not in any triangle, we know that $N(u_1) \setminus \{v\}$ is disjoint with $\{u_2, u_3, u_4\}$. For case (b2.2), we will generate $3 \times (d(u_1) - 1)$ subbranches by decreasing k by at least $2 + d(u_1) - 2 = d(u_1)$ in each, where $d(u_1) = 3$ or 4 .

The worst branching factors in the above steps are listed in Table 3. After Step *4, any degree-4 vertex can be only adjacent to vertices of degree at most 2. It is easy to see that the remaining graph after Step *4 is a proper graph. We call the algorithm in Lemma 15 to solve the instance for CO-PATH PACKING in $O^*(2.9241^k)$ time with probability at least $2/3$.

► **Theorem 17.** *CO-PATH PACKING can be solved in $O^*(2.9241^k)$ time with probability at least $2/3$.*

5 Conclusion

In this paper, we show that given a path decomposition of width p , CO-PATH PACKING can be solved by a randomized fpt algorithm running in $O^*(5^p)$ time. Additionally, by combining this algorithm with a branch-and-search algorithm, we show that CO-PATH/CYCLE PACKING can be solved in $O^*(2.8192^k)$ time and CO-PATH PACKING can be solved in $O^*(2.9241^k)$ time with probability at least $2/3$. For CO-PATH/CYCLE PACKING, the new bottleneck in our algorithm is Step 5, which is to deal with degree-4 vertices not in any triangle. For CO-PATH PACKING, the new bottleneck in our algorithm is the dynamic programming phase. The idea of using path/tree decomposition to avoid bottlenecks in branch-and-search algorithms may have the potential to be applied to more problems. It would also be interesting to design a deterministic algorithm for CO-PATH PACKING faster than $O^*(3^k)$.

References

- 1 Radovan Červený and Ondřej Suchý. Generating faster algorithms for d-path vertex cover. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 157–171. Springer, 2023.
- 2 Maw-Shang Chang, Li-Hsuan Chen, Ling-Ju Hung, Peter Rossmanith, and Ping-Chen Su. Fixed-parameter algorithms for vertex cover p3. *Discrete Optimization*, 19:12–22, 2016. doi:10.1016/J.DISOPT.2015.11.003.
- 3 Jianer Chen, Iyad A Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theoretical Computer Science*, 411(40-42):3736–3756, 2010. doi:10.1016/J.TCS.2010.06.026.
- 4 Zhi-Zhong Chen, Michael Fellows, Bin Fu, Haitao Jiang, Yang Liu, Lusheng Wang, and Binhai Zhu. A linear kernel for co-path/cycle packing. In *Algorithmic Aspects in Information and Management: 6th International Conference, AAIM 2010, Weihai, China, July 19-21, 2010. Proceedings 6*, pages 90–102. Springer, 2010. doi:10.1007/978-3-642-14355-7_10.
- 5 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5(4). Springer, 2015. doi:10.1007/978-3-319-21275-3.
- 6 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Joham MM van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 150–159. IEEE, 2011.

- 7 Michael R Fellows, Jiong Guo, Hannes Moser, and Rolf Niedermeier. A generalization of nemhauser and trotter's local optimization theorem. *Journal of Computer and System Sciences*, 77(6):1141–1158, 2011. doi:10.1016/J.JCSS.2010.12.001.
- 8 Qilong Feng, Jianxin Wang, Shaohua Li, and Jianer Chen. Randomized parameterized algorithms for p_2 -packing and co-path packing problems. *Journal of Combinatorial Optimization*, 29:125–140, 2015. doi:10.1007/S10878-013-9691-Z.
- 9 Fedor V Fomin, Serge Gaspers, Daniel Lokshtanov, and Saket Saurabh. Exact algorithms via monotone local search. *Journal of the ACM (JACM)*, 66(2):1–23, 2019. doi:10.1145/3284176.
- 10 Fedor V Fomin, Serge Gaspers, Saket Saurabh, and Alexey A Stepanov. On two techniques of combining branching and treewidth. *Algorithmica*, 54:181–207, 2009. doi:10.1007/S00453-007-9133-3.
- 11 David G Harris and NS Narayanaswamy. A faster algorithm for vertex cover parameterized by solution size. *arXiv preprint*, 2022. arXiv:2205.08022.
- 12 Ján Katrenič. A faster fpt algorithm for 3-path vertex cover. *Information Processing Letters*, 116(4):273–278, 2016. doi:10.1016/J.IPL.2015.12.002.
- 13 Dieter Kratsch and FV Fomin. *Exact exponential algorithms*. Springer, 2010.
- 14 Michael Lampis and Manolis Vasilakis. Structural parameterizations for two bounded degree problems revisited. *ACM Transactions on Computation Theory*, 2023.
- 15 Yuxi Liu and Mingyu Xiao. Solving co-path/cycle packing and co-path packing faster than 3^k . *arXiv preprint*, 2024. arXiv:2406.10829.
- 16 Stephen B Seidman and Brian L Foster. A graph-theoretic generalization of the clique concept. *Journal of Mathematical sociology*, 6(1):139–154, 1978.
- 17 Blair D Sullivan and Andrew van der Poel. A fast parameterized algorithm for co-path set. *arXiv preprint*, 2016. arXiv:1603.04376.
- 18 Dekel Tsur. Parameterized algorithm for 3-path vertex cover. *Theoretical Computer Science*, 783:1–8, 2019. doi:10.1016/J.TCS.2019.03.013.
- 19 Dekel Tsur. Faster deterministic algorithms for co-path packing and co-path/cycle packing. *Journal of Combinatorial Optimization*, 44(5):3701–3710, 2022. doi:10.1007/S10878-022-00917-3.
- 20 Jianhua Tu. A fixed-parameter algorithm for the vertex cover p_3 problem. *Information Processing Letters*, 115(2):96–99, 2015. doi:10.1016/J.IPL.2014.06.018.
- 21 Johan MM van Rooij. A generic convolution algorithm for join operations on tree decompositions. In *Computer Science—Theory and Applications: 16th International Computer Science Symposium in Russia, CSR 2021, Sochi, Russia, June 28–July 2, 2021, Proceedings 16*, pages 435–459. Springer, 2021.
- 22 Mingyu Xiao. A parameterized algorithm for bounded-degree vertex deletion. In *International Computing and Combinatorics Conference*, pages 79–91. Springer, 2016. doi:10.1007/978-3-319-42634-1_7.
- 23 Mingyu Xiao. On a generalization of Nemhauser and Trotter's local optimization theorem. *Journal of Computer and System Sciences*, 84:97–106, 2017. doi:10.1016/J.JCSS.2016.08.003.
- 24 Mingyu Xiao and Shaowei Kou. Kernelization and parameterized algorithms for 3-path vertex cover. In *International Conference on Theory and Applications of Models of Computation*, pages 654–668. Springer, 2017. doi:10.1007/978-3-319-55911-7_47.

A Polynomial Time Algorithm for Steiner Tree When Terminals Avoid a Rooted K_4 -Minor

Carla Groenland  

Delft Institute of Applied Mathematics, The Netherlands

Jesper Nederlof  

Department of Information and Computing Sciences, Utrecht University, The Netherlands

Tomohiro Koana  

Department of Information and Computing Sciences, Utrecht University, The Netherlands

Abstract

We study a special case of the Steiner Tree problem in which the input graph does not have a minor model of a complete graph on 4 vertices for which all branch sets contain a terminal. We show that this problem can be solved in $O(n^4)$ time, where n denotes the number of vertices in the input graph. This generalizes a seminal paper by Erickson et al. [Math. Oper. Res., 1987] that solves Steiner tree on planar graphs with all terminals on one face in polynomial time.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Steiner tree, rooted minor

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.12

Related Version *ArXiv Version*: <https://arxiv.org/abs/2410.06793>

Funding The work of both JN and TK has been supported by the project CRACKNP that has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No 853234).

1 Introduction

Planar graphs are well-studied algorithmically. For example, starting with the work of Baker [1], many efficient approximation schemes for NP-complete problems on planar graphs have been designed. Within parameterized complexity, widely applicable tools such as bi-dimensionality [6] helped to grasp a firm understanding of the “square-root phenomenon”: Many problems can be solved in $2^{O(\sqrt{n})}$ time, or even in $2^{O(\sqrt{k})} \cdot n^{O(1)}$ time, where n denotes the number of vertices of the input graph and k denotes the size of the sought solution. Actually most of these algorithms are also shown to extend to superclasses¹ of planar graphs, such as bounded genus or even minor free graphs (as for example showed in [6]).

One problem that is very well-studied on planar graphs is STEINER TREE. In the STEINER TREE problem we are given a weighted graph $G = (V, E)$ on n vertices with edge weights $w_e \in \mathbb{R}_{\geq 0}$ for $e \in E$ and a set of vertices $T = \{t_1, \dots, t_k\} \subseteq V$ called *terminals*, and we are tasked with finding an edge set S minimizing $w(S) = \sum_{e \in S} w(e)$ such that any pair of terminals t, t' are connected in the subgraph (V, S) . For example, an efficient approximation scheme was given in [4], and a lower bound excluding $2^{o(k)}$ time algorithms on planar graphs under the Exponential Time Hypothesis was given in [17]. Interestingly, the latter result shows that planarity alone is not too helpful to solve STEINER TREE quickly, because it implies that the classic $3^k n^{O(1)}$ dynamic programming algorithm for general STEINER TREE [8] cannot be significantly improved.

¹ Wagner’s theorem states that a graph is planar unless it contains a complete graph on 5 vertices (K_5) or complete bipartite graph with two blocks of 3 vertices ($K_{3,3}$) as a minor.



A seminal paper by Erickson et al. [9] shows that STEINER TREE on planar graphs is in P if all terminals lie on one face in a planar embedding of G . The study of the setting in which terminals lie in a few number of faces dates back all the way to the work of Ford and Fulkerson [11], and has been the subject of many classic works (such as the Okamura-Seymour Theorem [21, Chapter 74] or [18]). The algorithm [9] is used as subroutine in several other papers such as the aforementioned approximation scheme [4], but also preprocessing algorithms [19].

Often the algorithms that exploit planarity or minor-freeness need to combine *graph theoretic* techniques (such as a grid minor theorem in the case of bi-dimensionality) with *algorithmic* perspective (such as divide and conquer or dynamic programming over tree decompositions). For the STEINER TREE problem, and especially the algorithm from [9], the graph theoretic techniques employed are intrinsically *geometric*. And indeed, many extensions of the algorithm, such as the one in which the terminals can be covered the k outer faces of the graph [2], or the setting in which the terminals can be covered by a “path-convex region” studied in [20], all have a strong geometric flavor.

With this in mind, it is natural to ask whether there is an extension of the algorithm of [9] to minor free graphs.

Rooted Minors

We study a setting with *rooted* minors. A graph H is a *minor* of a graph G if it can be constructed from G by removing vertices or edges, or contracting edges. In a more general setting, G has a set of *roots* $R \subseteq V(G)$ and there is a mapping π that prescribes for each $v \in V(H)$ the set of roots $\pi(v) \subseteq R$. Then a *rooted minor* is a minor that contracts r into v , for every $v \in V(H)$ and $r \in R$ such that $r \in \pi(v)$. Rooted minors play a central role in Robertson and Seymour’s graph minor theory, and directly generalize the DISJOINT PATHS problem. Recently it was shown [16] that rooted minors can be detected in $(m+n)^{1+o(1)}$ time, for fixed $|H|$ and $|R|$, improving over the quadratic time algorithm from [15]. A number of recent papers have studied rooted minors in their own right [3, 12–14, 22, 23]. In particular, Fabila-Monroy and Wood [10] gave a characterization of when a graph contains a K_4 -minor rooted at four given vertices. Recently, links between rooted minors and “rooted” variants of treewidth, pathwidth and treedepth were given in [12].

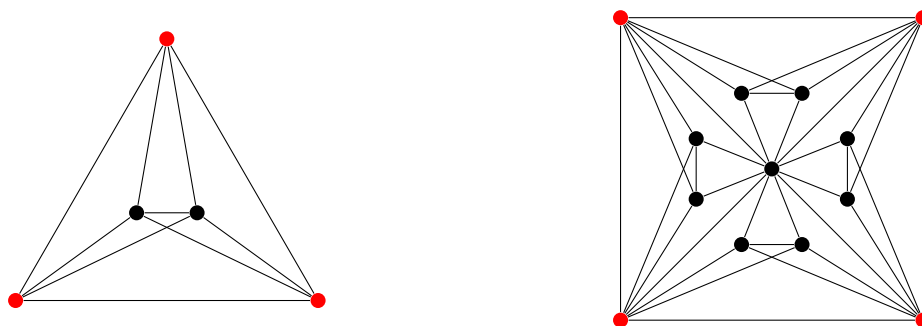
Motivated by the missing extension of the algorithm of [9] to a (non-geometric) setting of excluded minors, we propose studying the complexity of instances without minors rooted at the terminals.

Our Result

We will be interested in the closely related (but slightly different) setting of *R -rooted K_4 -minor*: In a graph $G = (V, E)$ with vertices $R \subseteq V$ (referred to as *roots*), an *R -rooted K_4 -minor* is a collection of disjoint vertex sets $S_1, S_2, S_3, S_4 \subseteq V$ such that $G[S_i]$ is connected and $S_i \cap R \neq \emptyset$ for all $i \in \{1, \dots, 4\}$, and such that there is an edge from S_i to S_j for each $i \neq j$.

► **Theorem 1.** *STEINER TREE without K_4 minor rooted at terminals can be solved in $O(n^4)$ time.*

This generalizes the result from [9]: It is easily seen that a planar graph has no K_4 -minor rooted at four vertices on the same face. On the other hand, it is easy to come up with example instances that have no K_4 -minor and are not planar (see Figure 1 for such two such examples).



■ **Figure 1** Terminals are depicted in red. The left figure has 3 terminals and hence no rooted K_4 minor, but it is a K_5 and hence not planar. The right figure has no rooted K_4 minor (since the middle vertex and two non-adjacent terminals separate the remaining terminals), and has many K_5 subgraphs and hence is not planar.

We hope that our result paves the road for additional polynomial time algorithms (solving non-geometrically) restricted versions of STEINER TREE. In particular, it would be interesting to see whether Theorem 1 can be extended to a polynomial time algorithm for a richer set of rooted \mathcal{F} -minor free instances (e.g., instances that do not contain any member of \mathcal{F} as rooted minor). Note however that such a strengthening with $\mathcal{F} = \{K_5\}$ would imply P=NP since STEINER TREE on planar graphs is NP-hard.

Our Approach

It is not too difficult to show from ideas given in [10] that if there is no R -rooted K_4 -minor in a 3-connected graph, then there is a cycle C passing through all vertices in R . We need a slightly stronger variant of this result, where the graph is not quite 3-connected but the only 2-cuts allowed are those which isolate a single vertex that is a terminal (see Lemma 4). Our algorithm will recurse on 2-cuts until it can apply this lemma, and then performs dynamic programming in a similar fashion to the Dreyfus-Wagner algorithm [8] restricted to subsets of terminals that form a contiguous segment of the cycle C (in a fashion that is analogous to the algorithm for [9], although the virtual edges considerably increase the technical difficulty of our proof). Since the number of such segments is at most quadratic in n , this gives a polynomial time algorithm.

We stress that the approach of [9] does not work directly: The approach of [9] crucially relies on the fact that if one decomposes an optimal Steiner Tree S into two trees $S_1 \cup S_2$ where S_1, S_2 are the connected components of $S \setminus e$ for some $e \in S$, then the set R_1 (and R_2) of terminals covered by S_1 (and S_2) is a contiguously visited along the cycle enclosing the face that contains all terminals. In our setting, there is a priori no guarantee how the set R_1 will look, and therefore such a decomposition approach will not work. To overcome this, we also decompose the optimal tree along 2-cuts via recursion and processing outcomes of recursive calls with virtual edges. Even though the virtual edges lead to some technical challenges, this allows us to make the idea of [9] work in 3-connected graphs.

2 Preliminaries

For $n \in \mathbb{N}$, we define $[n]$ as the set $\{1, \dots, n\}$. In this work, we assume that all graphs do not have self-loops, but we allow the graph to have parallel edges. We will say that a graph is *simple* if there is no parallel edge. The vertex set and edge set of a graph G are denoted by $V(G)$ and $E(G)$, respectively.

For two vertices u and v , let $\text{dist}(u, v)$ be the length of a shortest path between u and v . Let $X \subseteq V(G)$ be a subset of vertices. We use $G[X]$ to denote the subgraph induced by X and $G - X := G[V(G) \setminus X]$ to represent the graph obtained by removing the vertices in X . For these notations, if X is a singleton $\{x\}$, we may write x instead of $\{x\}$. Moreover, X is called a *cut* if deleting X increases the number of connected components. In particular, x is called a *cut vertex* if $\{x\}$ is a cut. For an edge set $F \subseteq E$, let $V(F)$ denote the set of vertices covered by F , i.e., $V(F) = \{u \mid u \in e \in F\}$.

Avoiding a (rooted) minor is preserved under deletion of vertices, deletion of edges and contraction of edges.

► **Observation 2.** *If G does not contain an R -rooted K_4 -minor and H is a minor of G , then H does not contain an $(V(H) \cap R)$ -rooted K_4 -minor.*

Our structural analysis will also crucially build on the following simple lemma.

► **Lemma 3** (Lemma 7 in [10]). *Suppose that a graph G has a cycle containing vertices v_1, v_2, v_3, v_4 in that order. Suppose moreover that there are two disjoint paths P_1 and P_2 where P_1 is from v_1 to v_3 , and P_2 is from v_2 to v_4 . Then G has a $\{v_1, v_2, v_3, v_4\}$ -rooted K_4 -minor.*

3 Finding a cycle passing through terminals and virtual edges

In this section, we prove a structural lemma that our algorithm needs. This builds on ideas from [10], but we require additional analysis due to the presence of “virtual edges”.

► **Lemma 4.** *Let $G = (V, E)$ be a 3-connected graph and let $R \subseteq V$ be a set of roots with $|R| \geq 3$. Let $E' \subseteq E$ and let G' be obtained by subdividing each edge in E' once. Let S denote the set of subdivision vertices added with this operation.*

If G' has no $(R \cup S)$ -rooted K_4 -minor, then G' contains a cycle containing all vertices in $R \cup S$. Furthermore, this cycle can be found in $nm^{1+o(1)}$ time for n the number of vertices and m the number of edges of G' .

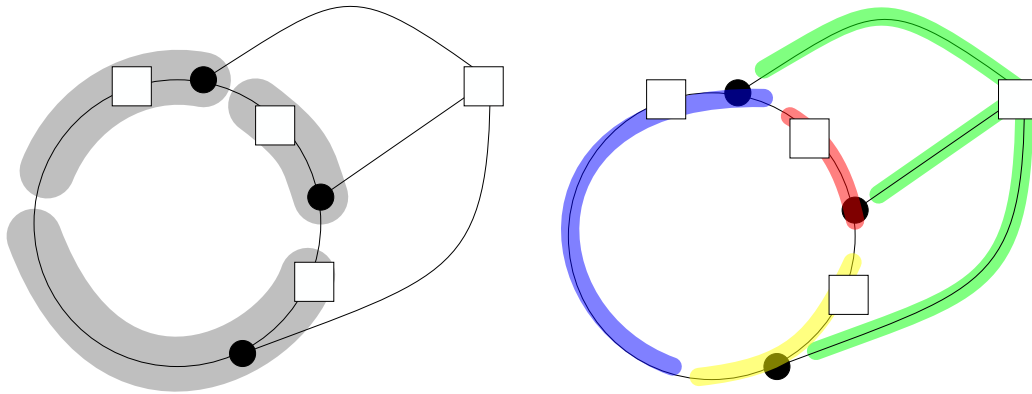
Proof. We first show that if there is no R -rooted K_4 -minor in G and G is 3-connected, then we can find a cycle C in G that passes through all vertices of R .

We start with C being any cycle. Suppose there is a vertex $r \in R$ that is not on C . Since G is 3-connected, by Menger’s theorem (see [7, Theorem 3.3.1], applied with sets C and $N(r)$), there exist three paths P_1, P_2, P_3 from r to C , where the paths mutually only intersect in r and each path only intersects C in their other endpoint. We can find these paths in $m^{1+o(1)}$ time with the max flow algorithm from e.g. [5]. Let $v_1, v_2, v_3 \in C$ denote these endpoints. If there are three disjoint arcs on the cycle that each contains a root from R and one of $\{v_1, v_2, v_3\}$, then G contains a rooted K_4 minor (see Figure 2).

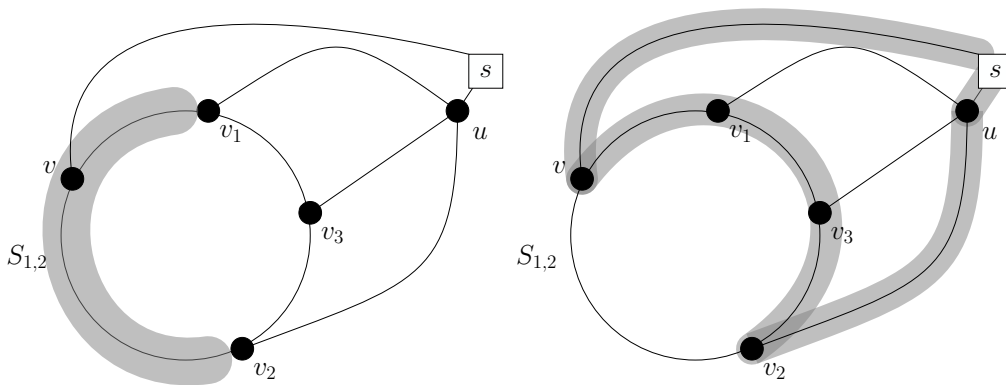
But if such arcs do not exist, then after renumbering we may assume that there is no root vertex between v_1 and v_2 on C . The cycle C' which goes from v_1 to v_2 via the path contained in C containing v_3 , and then back to v_1 via r via P_1 and P_3 , forms a cycle containing $(R \cap V(C)) \cup \{r\}$. Hence we can reiterate with this cycle until C contains R .

This means there is also a cycle C in G' containing all vertices in R : whenever an edge in E' is used, it is possible to pass through the subdivision vertex instead. We next modify the cycle C to be a cycle containing all vertices in R and all vertices in S .

Let $uv \in E'$ be the edge whose subdivision resulted in s . We first ensure that C contains both u and v (and R and all vertices in S already contained in it).



■ **Figure 2** If there are three vertex-disjoint paths from a root to disjoint arcs of the cycle containing roots (left), we obtain a rooted K_4 -minor (right). The roots are depicted by boxes.

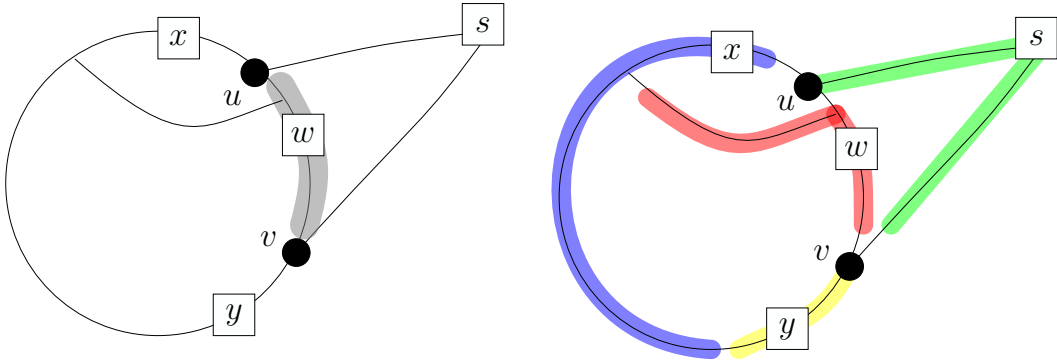


■ **Figure 3** If the segment $S_{1,2}$ contains v and no roots of C (left), then there is a cycle through v, u, s and all roots of C .

Suppose that u is not in C . As before, since G is 3-connected and $u \in V(G)$, there must be three paths from u to C , vertex-disjoint except for their endpoint u , meeting C in distinct endpoints v_1, v_2, v_3 . Again, we can find these paths in $m^{1+o(1)}$ time with [5]. We consider the same two cases as before.

- *Case 1: there are three disjoint arcs on C , each of which contains a vertex of $R \cup S$ and one vertex from $\{v_1, v_2, v_3\}$.* After contracting the edge between s and u , we obtain a rooted K_4 -minor in the same way as in the 3-connected case (see Figure 2). Hence this case does not happen by assumption.
- *Case 2: the segment $S_{1,2}$ between v_1 and v_2 (excluding v_1, v_2 and v_3) contains no vertices from $S \cup R$.* Then there is a cycle C' containing $V(C) \cap (S \cup R)$ and u . Note that C' no longer contains the vertices from $S_{1,2}$, so we could potentially lose v if $v \in S_{1,2}$. But if $v \in S_{1,2}$, then there is a cycle C'' that goes through $V(C) \cap (S \cup R) \cup \{s\}$ (see Figure 3).

Either way we ensured that the cycle contains u without affecting whether it contains v . After renumbering if needed, we are always either in Case 1 or Case 2. Repeating the argument above for v if needed, we can find a cycle passing through $(R \cup S) \cap V(C) \cup \{u, v\}$.



■ **Figure 4** The segment S_{uv} contains vertices between u and v , including w and excluding u and v itself (left). If there is a path from S_{uv} to x or the segment between x and y , then there is a rooted K_4 -minor (right), where v is included with y and u with s .

So we will assume C passes through u and v . Now we claim that one of the two arcs of C between $u, v \in C$ does not contain any vertices from $S \cup R$, and hence we may replace this by the path via s to get a cycle C that additionally passes through s .

Suppose this is not the case. Recall we assumed in our lemma statement that there are at least three vertices in R , and already established that C contains all vertices of R . So we can find w, x, y in $R \cup S$ such that C passes in order through u, w, v, y, x, u (with some vertices in between those, possibly). We choose x, y such that there are no vertices from $R \cup S$ between v and y and between x and u . (See also Figure 4.)

Let S_{uv} be the maximum segment of C between u and v which includes w but excludes u and v . That is, if $u, a_1, \dots, a_\ell, w, b_1, \dots, b_k, v$ are the vertices of C between u and v containing w , then

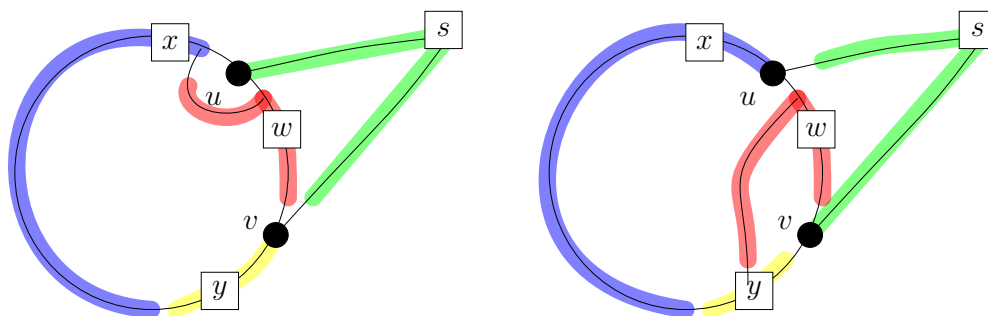
$$S_{uv} = \{a_1, \dots, a_\ell, w, b_1, \dots, b_k\}.$$

If there is a path intersecting C only in its endpoints from a vertex in S_{uv} to x or the segment of C in between x, y that excludes S_{uv} , then we have an $(S \cup R)$ -rooted K_4 -minor in G' , as shown in Figure 4.

Similarly, if there is a path from S_{uv} to y , a vertex in the segment between x and u (not including u) or the segment between y and v (not including v), then there is an $(S \cup R)$ -rooted K_4 -minor in G' , as depicted in Figure 5.

Since w and x are not the subdivision vertex of u and v , S_{uv} and $V(C) \setminus (\{u, v\} \cup S_{uv})$ both contain at least one vertex from G . Therefore, there is a path in $G \setminus \{u, v\}$ from $S_{uv} \cap V(G)$ to $(C \setminus S_{uv}) \cap V(G)$, since G is 3-connected. This means that one of the paths mentioned above exists, contradicting the fact that G' has no $(S \cup R)$ -rooted K_4 -minor. This contradiction shows that C must contain all vertices of S , as desired.

Hence, one of the two arcs of C between $u, v \in C$ does not contain any vertices from $S \cup R$, and we may replace this by the path via s to get a cycle C that additionally passes through s . Reiterating this argument at most n times gives the cycle passing through $R \cup S$. Since each iteration takes at most $m^{1+o(1)}$ time, the lemma follows. ◀



■ **Figure 5** If there is a path from S_{uv} to the segment between x and u (not including u), then there is a rooted K_4 -minor (left). If there is a path from S_{uv} to y , then there is a rooted K_4 -minor (right).

4 Description of algorithm

This section is organized as follows. In Section 4.1, we define an auxiliary problem called VIRTUAL EDGE STEINER TREE. In Section 4.2 we present a dynamic programming algorithm for VIRTUAL EDGE STEINER TREE for the case that all roots lie on a cycle. This cycle will be provided to us by Lemma 4. Section 4.3 discusses preprocessing steps. Finally, in Section 4.4, we describe our algorithm for VIRTUAL EDGE STEINER TREE.

4.1 Virtual edges

We first slightly generalize the STEINER TREE problem to a problem that we call VIRTUAL EDGE STEINER TREE in order to facilitate a recursive approach.

This problem is defined as follows. The input to VIRTUAL EDGE STEINER TREE is

- a graph $G = (V, E)$ with weights w_e for $e \in E$,
- a set of terminals $T = \{t_1, \dots, t_k\} \subseteq V$,
- a set of “virtual edges” E^* with weights $w_{e^*} : \{u, v, d, c\} \rightarrow \mathbb{R}_{\geq 0}$ for $e^* = \{u, v\} \in E^*$.

We will assume that $E \cap E^* = \emptyset$.

A *solution* to the VIRTUAL EDGE STEINER TREE problem is a (virtual) edge set $S \subseteq E \cup E^*$ such that $T \subseteq V(S)$ and $V(S) \cap V(e^*) \neq \emptyset$ for each virtual edge $e^* \in E$. Since the edge weights are non-negative we can assume S is a tree. For a (virtual) edge set S , the *cost* $c_{e^*}(S)$ of a virtual edge $e^* = \{u, v\} \in E^*$ with respect to S is given by

- $w_{e^*}(u)$ if $u \in V(S)$ and $v \notin V(S)$, representing the cost when u is included in the solution and v is not,
- $w_{e^*}(v)$ if $v \in V(S)$ and $u \notin V(S)$, representing the cost when v is included in the solution and u is not,
- $w_{e^*}(c)$ if $e \in S$, representing the cost when u, v are both included in the solution and connected “via the virtual edge”, that is, in the part of the graph we “forgot about”,
- $w_{e^*}(d)$ if $u, v \in V(S)$ and $e \notin S$, representing the cost when u, v are both included in the solution but are not connected “via the virtual edge”.

The *total cost* of a solution S is defined as

$$c(S) = \sum_{e \in S} w_e + \sum_{e^* \in E^*} c_{e^*}(S). \quad (1)$$

The purpose of virtual edges with cost functions is to enable our algorithm to handle 2-cuts $\{u, v\}$ effectively. When the algorithm identifies such a cut, it recursively solves the four subproblems for the smaller side first, encoding the costs in $w_{\{u,v\}}$. It is crucial to ensure that the solutions on both sides of the cut agree on the inclusion of vertices u and v in the final solution. Additionally, if both vertices are included, the algorithm must determine which side will contain the path connecting them.

In an instance of VIRTUAL EDGE STEINER TREE, we will assume that there is no terminal incident with a virtual edge. If there is a terminal t and a virtual edge tt' , then we assign the virtual edge weight $w_{tt'}(t') = \infty$, and remove t from the terminal set.

It will be convenient for the description of the dynamic program to view a root as either a singleton set consisting of a vertex (if it is a terminal), or an edge (if it is a virtual edge). Therefore, for an instance $(G, w, T, E^*, \{w_{e^*}\}_{e^* \in E^*})$, we define the set of roots as $R = \{\{t\} : t \in T\} \cup E^*$. We say the instance has no rooted K_4 -minor if the graph G^* has no R^* -rooted K_4 -minor, where G^* is the graph obtained from the graph $(V, E \cup E^*)$ by subdividing each edge in E^* once, and R^* is the union of T and the subdivision vertices of G^* .

Note that VIRTUAL EDGE STEINER TREE can be reduced to STEINER TREE as follows:

► **Observation 5.** *There is a reduction from an instance of VIRTUAL EDGE STEINER TREE with k terminals and ℓ virtual edges to 4^ℓ instances of STEINER TREE with at most $k + 2\ell$ terminals.*

Proof. Recall that there are four cases for each virtual edge: (i) u is covered but v is not, (ii) v is covered but u is not, (iii) both u and v are covered and the edge uv is part of the solution, and (iv) u and v are covered and the edge uv is not part of the solution. For each virtual edge, we delete it from the graph, and do one of the following. For case (i), add u as a terminal and delete v . For case (ii), add v as a terminal and delete u . For case (iii), contract u and v into a single vertex and add it as a terminal. For case (iv), add both u and v as terminals. This results in a STEINER TREE instance with at most $k + 2\ell$ terminals. The minimum cost of solutions among these instances plus the virtual edge weights is the minimum cost of the original instance of VIRTUAL EDGE STEINER TREE. ◀

4.2 Dynamic programming when roots lie on a cycle

In this subsection, we present a polynomial-time algorithm for the VIRTUAL EDGE STEINER TREE problem when all roots lie on a cycle.

► **Lemma 6.** *An instance of VIRTUAL EDGE STEINER TREE on an n -vertex graph without rooted K_4 -minor that has a cycle passing through all roots, can be solved in $O(n^4)$ time.*

Proof. We may also assume that there are at least five roots, since otherwise the problem can be efficiently solved via Observation 5. Let $R = \{r_1, \dots, r_k\}$ denote the set of roots. We renumber the indices so that r_1, \dots, r_k appear on C in the order of their indices. Recall that no terminal is incident with a virtual edge, and since the virtual edges lie on C , each non-terminal is incident with at most two virtual edges. Indices will be considered modulo k (identifying r_k with r_0). We use the shorthand $R[a, b]$ to denote the set of roots $\{r_c \in R : c \in [a, b]\}$. In particular, $R[a, b] = \{r_a, \dots, r_k, r_1, \dots, r_b\}$ for $a > b$.

We say a tree $\mathcal{T} = (V(\mathcal{T}), E(\mathcal{T}))$ is a *partial interval solution* if $V(\mathcal{T}) \subseteq V(G)$, $E(\mathcal{T}) \subseteq E \cup E^*$ and the set $R(\mathcal{T})$ of roots $r \in R$ for which $r \cap V(\mathcal{T}) \neq \emptyset$ forms an interval $R[a, b]$ for some $a, b \in [k]$.

We determine the status of virtual edges incident with the partial interval solution in the same way as we do for a solution, and we also assign a cost in a way similar to (1), but we now ignore virtual edges that are not incident with S

$$c(\mathcal{T}) = \sum_{e \in E(\mathcal{T}) \cap E} w_e + \sum_{e^* \in E(\mathcal{T}) \cap E^*} c_{e^*}(E(\mathcal{T})).$$

Note that each solution $S \subseteq E \cup E^*$ gives rise to a partial interval solution $\mathcal{T} = (V(E) \cup V(E^*), E \cup E^*)$, since $R(\mathcal{T}) = [k]$ is always an interval.

We say a partial interval solution \mathcal{T} is a *minimal solution* if $R(\mathcal{T}) = R$ (so it contains all terminals and contains at least one endpoint per virtual edge) and it is minimal in the sense that there is no strict subtree which also has this property. The last assumption is needed for technical reasons since the weights could also be zero.

We compute a dynamic programming table DP with entries $\text{DP}[a, b, v, s_a, s_b]$, where $a, b \in [k]$, $v \in V(G) \setminus T$, $s_a \in r_a \cup \{\mathbf{d}, \mathbf{c}\}$ and $s_b \in r_b \cup \{\mathbf{d}, \mathbf{c}\}$. If r_a is a terminal, then the table may ignore s_a : we will store the same value irrespective s_a . The same applies to s_b .

The table entry $\text{DP}[a, b, v, s_a, s_b]$ represents the minimum cost of a partial interval solution \mathcal{T} with $R(\mathcal{T}) \supseteq R[a, b]$ and $v \in V(\mathcal{T})$, where the status of r_a and r_b are indicated by s_a and s_b , respectively. In fact, the minimum will not go over all partial interval solutions, but only those that can be built in a specific manner as defined next. This means the lower bound is always true.

We define a collection \mathcal{B} of partial interval solutions via the following set of rules.

- R1.** Every partial interval solution $(\{v\}, \emptyset)$ consisting of a single vertex with $v \in V(G)$ is in \mathcal{B} .
- R2.** Every partial interval solution \mathcal{T} obtained from a partial interval solution $\mathcal{T}' \in \mathcal{B}$ by adding a new vertex $v \in V(G)$ not incident to any roots with an edge in E is in \mathcal{B} .
- R3.** Every partial interval solution \mathcal{T} obtained from two partial interval solutions $\mathcal{T}_1, \mathcal{T}_2 \in \mathcal{B}$ by “gluing on v ” is in \mathcal{B} when $V(\mathcal{T}_1) \cap V(\mathcal{T}_2) = \{v\}$ and all points in the intersection of the intervals $R(\mathcal{T}_1)$ and $R(\mathcal{T}_2)$ are endpoints of both intervals.

Note that by the definition of partial interval solution, $R(\mathcal{T})$ is an interval $R[a, b]$ for each $\mathcal{T} \in \mathcal{B}$. The following claim shows our dynamic program may restrict itself to partial interval solutions.

▷ **Claim 7.** Let \mathcal{T} be a minimal solution. Then $\mathcal{T} \in \mathcal{B}$.

Proof. We want to root our tree \mathcal{T} in a well-chosen vertex v and eventually prove the claim by induction. For $w \in V(\mathcal{T})$, let \mathcal{T}_w denote the subtree of \mathcal{T} containing all descendants of w . The choice of v needs to ensure that $R(\mathcal{T}_w) \neq [k]$ for all children w of v . By minimality, $R(\mathcal{T}_w) \neq \emptyset$ for all children w of v .

If there is at least one terminal r_a , then we can choose $v = r_a$, since r_a will then be missing from $R(\mathcal{T}_w)$ for all $w \neq v$. We show in the remainder of this paragraph that such a vertex v can also be chosen when no terminal exists. If there is a virtual edge incident to a single vertex u , we can choose $v = u$ similar to the terminal case. Moreover, if there is a vertex u incident with two virtual edges, then we may choose $v = u$. Otherwise, all roots come from virtual edges that have both endpoints present in \mathcal{T} and each vertex is incident with at most one edge. We give each vertex of G incident with a virtual edge weight 1. Since \mathcal{T} is a tree, we can root it in a “balanced separator”: a vertex v such that the total weight in each component of $\mathcal{T} - v$ is at most half the total weight of \mathcal{T} . The only way $R(\mathcal{T}_w)$ can then be $[k]$ for w a child of v , is when all roots are coming from edges incident with one vertex in \mathcal{T}_w and one outside. Since there are at least five roots, we can choose roots r_a, r_b, r_c, r_d

consecutive on the cycle, and connect r_a to r_c using only internal vertices from $V(\mathcal{T}_w)$ and r_b to r_d using only internal vertices not in $V(\mathcal{T}_w)$, finding a rooted K_4 -minor with Lemma 3, a contradiction.

In this paragraph, we show that $R(\mathcal{T}_w)$ is an interval. We are done if $|R(\mathcal{T}_w)| \geq k - 1$. So we assume that $|R(\mathcal{T}_w)| \leq k - 2$ and $w \neq v$. Suppose towards a contradiction that $R(\mathcal{T}_w)$ is not an interval. This means there exist $a < b < c < d$ in $[k]$ such that $r_a, r_c \in R(\mathcal{T}_w)$ and $r_b, r_d \notin R(\mathcal{T}_w)$ (or $b < c < d < a$, but this case is analogous). This means there is a path between (the roots of) r_a and r_c via vertices in $V(\mathcal{T}_w)$. Moreover, r_b and r_d are not in $R(\mathcal{T}_w)$, and so they are either part of the tree \mathcal{T}' obtained from \mathcal{T} by removing \mathcal{T}_w , or incident to a vertex in this tree. This provides a path between (the roots of) r_b and r_d using vertices not in $V(\mathcal{T}_w) \cup R(\mathcal{T}_w)$. Thus, by Lemma 3, we obtain an $\{r_a, r_b, r_c, r_d\}$ -rooted K_4 -minor, a contradiction.

In this paragraph we show that $\mathcal{S}_w = (\{w\}, \emptyset) \in \mathcal{B}$ for each $w \in V(\mathcal{T})$. For this, we need to show that $R(\mathcal{T})$ forms an interval. Recall that no terminal is incident with a virtual edge, and since the virtual edges lie on C , each non-terminal is incident with at most two virtual edges which are then also consecutive on the cycle. This means $|\mathcal{S}_w| \leq 2$ and that $R(\mathcal{S}_w)$ is an interval. Since \mathcal{S}_w consists of a single vertex, it is now added to \mathcal{B} in **R1**.

The remainder of the proof shows that $\mathcal{T}_w \in \mathcal{B}$ for each $w \in V(\mathcal{T})$ by induction on $|V(\mathcal{T}_w)|$. We already proved (a stronger variant of) the case when the size is 1 above. Suppose $\mathcal{T}_w \in \mathcal{B}$ has been shown for all w with $|V(\mathcal{T}_w)| \leq \ell$ for some $\ell \geq 1$ and now assume $|V(\mathcal{T}_w)| = \ell + 1$.

We first show a property that we need in both cases considered below. Let w' be a child of w and suppose that $r_x \in R(\mathcal{S}_w) \cap R(\mathcal{T}_{w'})$. Then r_x must correspond to a virtual edge ww_1 with $w_1 \in V(\mathcal{T}_{w'})$. We show that r_x is an endpoint of the interval $R(\mathcal{T}_{w'})$. Recall that $R(\mathcal{T}_{w'}) \neq [k]$ by choice of v . If r_x is not an endpoint, then there are vertices $r_a, w_1, r_x, w, r_b, r_d$ appearing in order on the cycle for some $r_a, r_b \in R(\mathcal{T}_{w'})$ and $r_d \notin R(\mathcal{T}_{w'})$. In particular, r_x can be connected to r_d via w using vertices outside of $V(\mathcal{T}_{w'})$ and we can connect r_a and r_b using internal vertices in $V(\mathcal{T}_{w'})$. We then apply Lemma 3 to obtain an $\{r_a, r_x, r_b, r_d\}$ -rooted K_4 -minor, a contradiction.

We now turn to the proof that $\mathcal{T}_w \in \mathcal{B}$. We already proved its roots form an interval. Suppose that w has a single child w' . By the induction hypothesis, $\mathcal{T}_{w'}$ is a partial interval solution. If w is not incident with any roots, then \mathcal{T}_w can be obtained from $\mathcal{T}_{w'}$ with **R2**.

If w is incident with at least one root, we show we can obtain \mathcal{T}_w by gluing the partial interval solution $\mathcal{S}_w = (\{w\}, \emptyset)$ with $\mathcal{T}_{w'}$. To apply **R3**, we need to show all points in the intersection $R(\mathcal{S}_w) \cap R(\mathcal{T}_{w'})$ are endpoints of the corresponding intervals.

- If w is a terminal r_a , then it is not incident with any virtual edges and hence there are no intersection points.
- Suppose that w is not a terminal, but it is incident with either one or two virtual edges $rx_1 = ww_2$ and $rx_2 = ww_2$. This means $R(\mathcal{S}_w) = \{r_{x_1}, r_{x_2}\}$ and so these are the only possible intersection points. We proved above that if r_{x_j} exists and is in $R(\mathcal{T}_{w'})$, it must be an endpoint of that interval (for $j = 1, 2$). Moreover, each root in $R(\mathcal{S}_w)$ is automatically an endpoint of the interval since the size is 1 or 2.

So we may assume w has at least two children. Let w_1, \dots, w_d denote the children of w in \mathcal{T} and for each $i \in [d]$, let $\mathcal{T}_i = \mathcal{T}_{w_i}$ denote the subtree of \mathcal{T} containing all descendants of w_i . We already showed that for each $i \in [d]$, $R(\mathcal{T}_i)$ is an interval $R[a_i, b_i]$ for some $a_i, b_i \in [k]$.

The union of these intervals will be $R(\mathcal{T}_w) \setminus \{w\}$. We now describe when $R[a_i, b_i]$ and $R[a_j, b_j]$ can intersect for some $i \neq j$. Every intersection point comes from a virtual edge uu' with $u \in \mathcal{T}_i$ and $u' \in \mathcal{T}_j$, and in particular each root appears in at most two of the \mathcal{T}_i . We show that when $r_x \in R[a_i, b_i] \cap R[a_j, b_j]$, we must have $x \in \{a_i, b_i\}$. Indeed, if not, we select

$r_c \notin R[a_i, b_i]$ and now $r_c, r_{a_i}, r_x, r_{b_i}$ appear consecutively on the cycle, with a path between r_{a_i} and r_{b_i} contained within $V(\mathcal{T}_i)$ and a path between r_x and r_c outside of $V(\mathcal{T}_i)$ (via u'), yielding a contradiction via an $\{r_c, r_{a_i}, r_x, r_{b_i}\}$ -rooted K_4 -minor by Lemma 3.

Moreover, we already argued that $R(\mathcal{S}_w) \cap R[a_i, b_i]$ can only intersect in a_i or b_i (and in fact, at most one of those two by minimality arguments). Combined, this means that there is a way to renumber such that

$$\bigcup_{i=1}^j R[a_i, b_i] \cup R(\mathcal{S}_w) \text{ and } \bigcup_{i=j+1}^d R[a_i, b_i] \cup R(\mathcal{S}_w)$$

form intervals intersecting only in their endpoints. Here j is chosen so that either $w = r_j$ if w is terminal, and w is between r_{b_j} and $r_{a_{j+1}}$ on the cycle if one (or both) of those is a virtual edge incident to w , and otherwise $R(\mathcal{S}_w) = \emptyset$ and does not affect whether the sets are intervals. \triangleleft

Our dynamic program will ensure that each partial interval solution in \mathcal{B} is considered by following rules **R1-R3**. Since these rules are also easily seen to result in a valid Steiner tree, the final output therefore will be optimal by Claim 7.

Base case. All entries are set to ∞ initially. We compute all entries $\text{DP}[a, b, v, s_a, s_b]$ corresponding to a partial interval solution consisting of a single vertex via Observation 5. That is, for each vertex v ,

- if v is a terminal r_a , we set $\text{DP}[a, a, v, s_a, s'_a] = 0$;
- if v has exactly one incident virtual edge r_a , we set $\text{DP}[a, a, v, v, v] = w_{r_a}(v)$.
- if v has incident virtual edges r_a and r_{a+1} , we set $\text{DP}[a, a+1, v, v, v] = w_{r_a}(v) + w_{r_{a+1}}(v)$.

Adding a vertex. We first implement **R2**: adding a vertex not incident to any roots. Suppose that v is not incident with any roots. For any edge $uv \in E$, we add the rule

$$\text{DP}[a, b, v, s_a, s_b] \leftarrow \min(\text{DP}[a, b, u, s_a, s_b] + w_{uv}, \text{DP}[a, b, v, s_a, s_b]).$$

Next, we implement **R3**, “gluing on v ”, which has multiple cases because we also need to properly keep track of the costs of the virtual edges.

Gluing two solutions without intersection. We first consider the case in which the intervals of the roots are disjoint, in which case the partial interval solutions that we are gluing do not have any virtual edges between them. When $a_2 = b_1 + 1$ and $a_1 \notin [a_2, b_2]$, we set

$$\text{DP}[a_1, b_2, v, s_{a_1}, s_{b_2}] \leftarrow \min(\text{DP}[a_1, b_2, v, s_{a_1}, s_{b_2}], \text{DP}[a_1, b_2, v, s_{a_1}, s_{b_2}] + \text{DP}[a_2, b_2, v, s_{a_2}, s_{b_2}]).$$

Gluing two solutions with one intersection. Now suppose that the intervals overlap in exactly one point.

When $a_2 = b_1$ with r_{a_2} a terminal and $a_1 \notin [a_2 + 1, b_2]$, we set

$$\text{DP}[a_1, b_2, v, s_{a_1}, s_{b_2}] \leftarrow \min(\text{DP}[a_1, b_2, v, s_{a_1}, s_{b_2}], \text{DP}[a_1, b_1, v, s_{a_1}, s_{b_1}] + \text{DP}[a_2, b_2, v, s_{a_2}, s_{b_2}]).$$

We may forget about s_{a_2}, s_{b_1} since the “status” of the terminal is irrelevant to us.

When $r_{a_2} = uu'$ is a virtual edge, we need to consider various options, depending on the status this edge used to be in for both solutions, and what final state we want it to be, and whether $a_1 = b_1$ and/or $a_2 = b_2$.

12:12 A Polynomial Time Algorithm for Steiner Tree When Terminals Avoid a K_4 -Minor

We start with when we want to make the status d . We need to ensure u, u' are part of the solution, and the solutions must overlap elsewhere in some vertex v . So when $a_2 = b_1$, $a_1 \notin [a_2 + 1, b_2]$, we set

$$\begin{aligned} \text{DP}[a_1, b_2, v, x_1, x_2] \leftarrow & \min(\text{DP}[a_1, b_2, v, x_1, x_2], \\ & \text{DP}[a_1, b_1, v, s_{a_1}, u] + \text{DP}[a_2, b_2, v, u', s_{b_2}] - w_{uu'}(u) - w_{uu'}(u') + w_{uu'}(d), \\ & \text{DP}[a_1, b_1, v, s_{a_1}, u'] + \text{DP}[a_2, b_2, v, u, s_{b_2}] - w_{uu'}(u) - w_{uu'}(u') + w_{uu'}(d)), \end{aligned}$$

where we require $x_1 = s_{a_1}$ if $a_1 \neq b_1$ and $x_1 = d$ when $a_1 = b_1$; and we require $x_2 = s_{b_2}$ if $a_2 \neq b_2$ and $x_2 = d$ when $a_2 = b_2$. In the last line, for example, we use that the first partial interval solution has paid the cost $w_{uu'}(u')$ (as recorded by the status) and the second $w_{uu'}(u)$. We allow here to replace the cost by $w_{uu'}(d)$, since the solutions can be merged via the vertex v . If $a_1 = b_1$ or $a_2 = b_2$, we will keep the status d recorded at the relevant endpoint.

We also give the option to “connect” via the virtual edge r_{a_2} . When $r_{a_2} = uu'$ is a virtual edge, $a_2 = b_1$, $a_1 \notin [a_2 + 1, b_2]$, we set

$$\begin{aligned} \text{DP}[a_1, b_2, u, x_1, x_2] \leftarrow & \min(\text{DP}[a_1, b_2, u, x_1, x_2], \\ & \text{DP}[a_1, b_1, u, s_{a_1}, u] + \text{DP}[a_2, b_2, u', u', s_{b_2}] - w_{uu'}(u) - w_{uu'}(u') + w_{uu'}(c)), \\ \text{DP}[a_1, b_2, u', x_1, x_2] \leftarrow & \min(\text{DP}[a_1, b_2, u', x_1, x_2], \\ & \text{DP}[a_1, b_1, u, s_{a_1}, u] + \text{DP}[a_2, b_2, u', u', s_{b_2}] - w_{uu'}(u) - w_{uu'}(u') + w_{uu'}(c)), \end{aligned}$$

where again, we require $x_i = s_{a_i}$ if $a_i \neq b_i$ and $x_i = c$ otherwise.

To allow the edge to be in status u (or u' , analogously), we also add

$$\begin{aligned} \text{DP}[a_1, b_2, v, x_1, x_2] \leftarrow & \min(\text{DP}[a_1, b_2, v, x_1, x_2], \\ & \text{DP}[a_1, b_1, v, s_{a_1}, u'] + \text{DP}[a_2, b_2, v, u, s_{b_2}] - w_{uu'}(u'), \\ & \text{DP}[a_1, b_1, v, s_{a_1}, u] + \text{DP}[a_2, b_2, v, u, s_{b_2}] - w_{uu'}(u), \\ & \text{DP}[a_1, b_1, v, s_{a_1}, u] + \text{DP}[a_2, b_2, v, u', s_{b_2}] - w_{uu'}(u')), \end{aligned}$$

where again, we require $x_i = s_{a_i}$ if $a_i \neq b_i$ and $x_i = c$ otherwise.

Gluing two solutions with two intersections It remains now to glue intervals with two intersection points to a final solution. The rules for this are analogous to the previous case, but now the compatibility is checked at both endpoints, $a_1 = b_2$ and $a_2 = b_1$.

We apply the rules in a bottom-up fashion. The final output is the minimum over all entries $\text{DP}[a, b, v, s_a, s_b]$ with $R[a, b] = R$.

Running time. There are $O(n^3)$ entries, each of which takes $O(n)$ time to compute. Thus, the total running time is $O(n^4)$. \blacktriangleleft

4.3 Preprocessing

Throughout, the algorithm works on a simple graph G^* (that is, at most one edge per pair of vertices). We can always obtain this via the following *edge pruning* steps.

1. If there are two edges e and e' containing the same two endpoints with $w_e \leq w_{e'}$, then delete e' .

2. If there are two virtual edges e_1^* and e_2^* over the same endpoints u and v , then delete e_1^* and e_2^* add a new virtual edge e^* between u and v with the virtual weight defined as follows:
 - $w_{e^*}(u) = w_{e_1^*}(u) + w_{e_2^*}(u)$.
 - $w_{e^*}(v) = w_{e_1^*}(v) + w_{e_2^*}(v)$.
 - $w_{e^*}(c) = \min(w_{e_1^*}(c) + w_{e_2^*}(d), w_{e_1^*}(d) + w_{e_2^*}(c))$.
 - $w_{e^*}(d) = w_{e_1^*}(d) + w_{e_2^*}(d)$.
3. If there is an edge e and virtual edge e^* over the same endpoints say u and v , then delete e , and update the virtual weight of e^* by $w_{e^*}(c) = \min(w_{e^*}(c), w_{e^*}(d) + w_e)$.

We briefly discuss the correctness of the preprocessing steps above, though all of them follow directly from the definition of VIRTUAL EDGE STEINER TREE. Firstly, note that we exactly remove all multiple edges (and preserve which vertices have at least one edge between them). For (1), an edge with a larger weight is never included in an optimal solution so can be removed. For (2), at most one of the two virtual edges will be used to “connect” the vertices, but we still need to pay the cost for both even if they are not “included” in the solution. Hence the new edge has the weight defined as the sum of the weight of two virtual edges for the cases u , v , and d . For the case c , we may assume that exactly one of e_1^* and e_2^* is included into the solution. Lastly, for (3), we update w_{e^*} to take into account that it may be cheaper to connect via the edge e in the scenario that u and v needs to be connected. That is, if there is an edge e and virtual edge e^* such that $w_{e^*}(c) \geq w_{e^*}(d) + w_e$, then we update $w_{e^*}(c)$ to $w_{e^*}(d) + w_e$. In the other scenarios an optimal solution would never add the edge e .

We define a preprocessing procedure as follows to ensure that every biconnected component has at least one root and that every triconnected component has at least two roots.

We say a vertex set $A \subseteq V$ is *incident with a root* if $A \cap T \neq \emptyset$ or there is a virtual edge incident with a vertex of A .

1. Perform edge pruning.
2. If there is a cut vertex v such that $G - v$ has a connected component A that is not incident with any roots, then delete A from the graph. Return to 1.
3. If there is a 2-cut $\{u, v\}$ such that $G[V \setminus \{u, v\}]$ has a connected component A that is not incident with a root, then delete A from the graph and add an edge uv , where the weight w_{uv} equals $\text{dist}_{G[A \cup \{u, v\}]}(u, v)$. Return to 1.
4. If there is a 2-cut $\{u, v\}$ such that $G[V \setminus \{u, v\}]$ has a connected component A that is incident with exactly one root, we compute the weights for a new virtual edge $\{u, v\}$ by solving four VIRTUAL EDGE STEINER TREE instances using Observation 5.
 - $w_{uv}(u)$ is the cost of the instance induced by $A \cup \{u\}$ with u as an additional terminal.
 - $w_{uv}(v)$ is the cost of the instance induced by $A \cup \{v\}$ with v as an additional terminal.
 - $w_{uv}(c)$ is the cost of the instance induced by $A \cup \{u, v\}$ with $\{u, v\}$ as terminals but where we do not include a virtual edge between u and v if it had one.
 - $w_{uv}(d) = \min(w_{uv}(u), w_{uv}(v))$.

We add a new virtual edge $\{u, v\}$ with the costs as defined above and remove all vertices from A . (If there was already a virtual edge between u and v , we keep it and it will be dealt with during the cleaning.) Return to 1.

For (2), note that since v is a cut vertex, there is always an optimal solution that has empty intersection with A .

For (3), if u and v are connected in an optimal solution via A , then this will be done via a shortest path between u and v .

■ **Algorithm 1** A polynomial-time algorithm for VIRTUAL EDGE STEINER TREE. We assume that the input graph is connected and that there is no rooted K_4 -minor in the instance.

```

1: procedure ALGO( $(G, T, E^*, w_e)$ )
2:   Apply preprocessing from Section 4.3
3:   if  $|T| + |E^*| = O(1)$  then return Solve by Observation 5 and Dreyfus-Wagner.
4:   if  $G^*$  is not 2-connected then
5:     Let  $v$  be a cut vertex that separates  $G$  into  $A$  and  $B$ .
6:     return ALGO( $G[A \cup \{v\}], (T \cap A) \cup \{v\}, E^* \cap \binom{A}{2}, w$ )
7:       + ALGO( $G[B \cup \{v\}], (T \cap B) \cup \{v\}, E^* \cap \binom{B}{2}, w$ ).
8:   else if  $G^*$  is not 3-connected then
9:     Let  $\{u, v\}$  be a cut that separates  $G$  into  $A$  and  $B$  with  $|A| \leq |B|$ .
10:    Let  $e$  be a virtual edge connecting  $u$  and  $v$ .
11:     $w_e(u) \leftarrow$  ALGO( $G[A \cup \{u\}], (T \cap A) \cup \{u\}, E^* \cap \binom{A}{2}, w$ )
12:     $w_e(v) \leftarrow$  ALGO( $G[A \cup \{v\}], (T \cap A) \cup \{v\}, E^* \cap \binom{A}{2}, w$ )
13:     $w_e(c) \leftarrow$  ALGO( $G[A \cup \{u, v\}], (T \cap A) \cup \{u, v\}, E^* \cap \binom{A}{2}, w$ )
14:     $w_e(d) \leftarrow$  ALGO( $G[A \cup \{u, v\}] + uv, (T \cap A) \cup \{u, v\}, E^* \cap \binom{A}{2}, w$ )
15:    return ALGO( $G[B \cup \{u, v\}], T, E^* \cup \{e\}, w$ )
16:   else
17:     return the result for 3-connected case.

```

For (4), there are four cases to consider. In the first three cases, we enforce u, v or both u and v are contained in the optimal Steiner tree by making them terminals. In the last, we note that the solution on A is allowed to go either via u or via v . Note that the instances that we solve to define the costs all have at most 3 roots (virtual edges or terminals) and so can be solved in polynomial time in terms of $|A|$ using Observation 5.

4.4 Our algorithm

See Algorithm 1 for the outline of the algorithm. Our algorithm maintains the invariant that there is no rooted K_4 -minor over the terminals T and virtual edges E^* . We first apply the preprocessing steps in Section 4.3. If this results in a graph with $O(1)$ roots, we solve the problem by reducing to STEINER TREE with $O(1)$ terminals, which can then be solved using the Dreyfus-Wagner algorithm [8] in $O(1)$ time.

G^* is not 2-connected. Let v be a cut vertex separating G into A and B . (Here, A and B each may contain multiple connected components of $G - v$.) We recursively solve two instances induced on vertex sets $A \cup \{v\}$ and $B \cup \{v\}$ where v is an additional terminal. To see why this recursion is correct, note that A and B each contains at least one terminal or virtual edge by the preprocessing steps. So, any Steiner tree must cover v as well, and thus it is obtained by “gluing” the two solutions on the vertex v . The total cost equals the sum of the two recursive instance costs.

We claim that no rooted K_4 -minor is introduced in the recursive calls. As v is a cut vertex, there is a path from v to each vertex in B . Moreover, by the preprocessing, B contains a root (there is either a virtual edge in $B \cup \{v\}$ or a terminal in B). This means that any rooted K_4 -minor in the instance on $A \cup \{v\}$ with v as terminal also gives rise to a rooted K_4 -minor in G .

G^* is not 3-connected. We first find a 2-cut $\{u, v\}$ that separates the graph into A and B with $|A| \leq |B|$. Our algorithm considers four cases based on whether u and v are covered by the solution.

- We solve the subproblem over $G[A \cup \{u, v\}]$ with u as an additional terminal, to account for the case in which u is covered but v is not.
- We solve the subproblem over $G[A \cup \{u, v\}]$ with v as an additional terminal, to account for the case in which v is covered but u is not.
- We solve the subproblem over $G[A \cup \{u, v\}]$ with u and v as additional terminals. This accounts for the case that both u and v are covered, and they are connected through A .
- We solve the subproblem over $G[A \cup \{u, v\}]$ with u as terminal and the edge $\{u, v\}$ added (in E) with cost $w(\{u, v\}) = 0$. This accounts for the case in which both u and v are covered, and they are not connected through A .

Next, we solve a single instance on vertex set $B \cup \{u, v\}$ where $\{u, v\}$ is added as virtual edge (in E^*). The cost of the virtual edge $\{u, v\}$ is determined by the previous calculations

We verify that no rooted K_4 -minor is introduced in the recursive calls. The first two recursive calls (corresponding to the cases u and v) are analogous to the scenario where G^* has a cut vertex. For the third and fourth calls (cases c and d), assume for contradiction that there exists a rooted K_4 -minor over $T \cup \{u, v\}$ in $G[A \cup \{u, v\}] + uv$. Our preprocessing steps ensure two roots $r, r' \in T \cup E^*$. Since there is no cut vertex, there are two disjoint paths connecting $\{r, r'\}$ to $\{u, v\}$. Using these paths, we can construct a rooted K_4 -minor in G , which is a contradiction. For the final recursive call on $B \cup \{u, v\}$, the argument for the case where G^* has a cut vertex again shows the non-existence of a rooted K_4 -minor.

G^* is 3-connected. If G^* is 3-connected, then we can find a polynomial time a cycle going through all roots by Lemma 4 presented in Section 3, and VIRTUAL EDGE STEINER TREE can be solved as described in Section 4.2

Runtime analysis

When we solve an instance with G^* being 3-connected, we already showed the running time is at most $c'n^4$ for some constant $c' > 0$, assuming $n \geq n_0$ for some constant n_0 . After making c' larger if needed, we can also assume all additional steps in the algorithm such as finding a 2-cut and preprocessing take at most $c'n^3$ on inputs of $n \geq n_0$ vertices.

We let $T(n)$ denote the maximal running time on an input graph on n vertices and show that $T(n) \leq cn^4$ for some constant $c > 0$ by induction on n . We will choose c such that $c \geq c'$ and $T(n) \leq cn^4$ when $n \leq n_0$.

When G^* is not 2-connected, we find a cut vertex v splitting the graph into A and B and recursively solve two instances on $A \cup \{v\}$ and $B \cup \{v\}$ and obtain the cost from there. So with $|A| = i$, we find the running time is at most $T(i+1) + T(n-i) + c'n^3$.

When G^* is 2-connected, but not 3-connected, we find a 2-cut $\{u, v\}$ that separates G^* into A and B with $|A| \leq |B|$, and therefore $|A| \leq (n-2)/2$. With $i = |A|$, we recursively solve four instances of size at most i and one instance of size $|B| \leq n-i$, resulting in a running time of at most

$$\max_{1 \leq i \leq n/2} T(n-i) + 4T(i+2) + c'n^3.$$

Applying the inductive hypothesis, we find $T(n-i) \leq c(n-i)^4$ and $T(i+2) \leq c(i+2)^4$ for all $1 \leq i \leq n/2$. The function $f(i) = c(n-i)^4 + 4c(i+2)^4 + c'n^3$ is convex within the domain

$1 \leq i \leq n/2$ because the second derivative $f''(i)$ (with respect to i) is positive. Therefore, the maximum value is attained either at $i = 1$ or $i = n/2$. Evaluating these points,

$$\begin{aligned} f(1) &= c(n-1)^4 + 4c(3^4) + c'n^3 \\ f(n/2) &= c(n/2)^4 + 4c(n/2 + 2)^4 + c'n^3, \end{aligned}$$

we see both are less than cn^4 when n is sufficiently large since $c \geq c'$. Hence, we conclude that $T(n) = O(n^4)$.

References

- 1 Brenda S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J. ACM*, 41(1):153–180, 1994. doi:10.1145/174644.174650.
- 2 Marshall W. Bern and Daniel Bienstock. Polynomially solvable special cases of the Steiner problem in planar networks. *Ann. Oper. Res.*, 33(6):403–418, 1991. doi:10.1007/BF02071979.
- 3 Thomas Böhme, Jochen Harant, Matthias Kriesell, Samuel Mohr, and Jens M. Schmidt. Rooted minors and locally spanning subgraphs. *J. Graph Theory*, 105(2):209–229, 2024. doi:10.1002/JGT.23012.
- 4 Glencora Borradaile, Philip N. Klein, and Claire Mathieu. An $O(n \log n)$ approximation scheme for Steiner tree in planar graphs. *ACM Trans. Algorithms*, 5(3):31:1–31:31, 2009. doi:10.1145/1541885.1541892.
- 5 Li Chen, Rasmus Kyng, Yang P. Liu, Simon Meierhans, and Maximilian Probst Gutenberg. Almost-linear time algorithms for incremental graphs: Cycle detection, sccs, s-t shortest path, and minimum-cost flow. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC*, pages 1165–1173. ACM, 2024. doi:10.1145/3618260.3649745.
- 6 Erik D. Demaine, Fedor V. Fomin, Mohammad Taghi Hajiaghayi, and Dimitrios M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H -minor-free graphs. *J. ACM*, 52(6):866–893, 2005. doi:10.1145/1101821.1101823.
- 7 R. Diestel. *Graph Theory: 5th edition*. Springer Graduate Texts in Mathematics. Springer Berlin, Heidelberg, 2017. URL: <https://books.google.nl/books?id=zIxRDwAAQBAJ>.
- 8 Stuart E. Dreyfus and Robert A. Wagner. The Steiner problem in graphs. *Networks*, 1(3):195–207, 1971. doi:10.1002/NET.3230010302.
- 9 Ranel E. Erickson, Clyde L. Monma, and Arthur F. Veinott Jr. Send-and-split method for minimum-concave-cost network flows. *Math. Oper. Res.*, 12(4):634–664, 1987. doi:10.1287/MOOR.12.4.634.
- 10 Ruy Fabila-Monroy and David R. Wood. Rooted K_4 -Minors. *Electronic Journal of Combinatorics*, 20(2):#P64, 2013. doi:10.37236/3476.
- 11 Lester Randolph Ford and Delbert R Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8:399–404, 1956.
- 12 Jędrzej Hodor, Hoang La, Piotr Micek, and Clément Rambaud. Quickly excluding an apex-forest, 2024.
- 13 Ken-ichi Kawarabayashi. Rooted minor problems in highly connected graphs. *Discrete Mathematics*, 287(1):121–123, 2004. doi:10.1016/j.disc.2004.07.007.
- 14 Leif K Jørgensen and Ken-ichi Kawarabayashi. Extremal results for rooted minor problems. *Journal of Graph Theory*, 55(3):191–207, 2007. doi:10.1002/jgt.20232.
- 15 Ken-ichi Kawarabayashi, Yusuke Kobayashi, and Bruce A. Reed. The disjoint paths problem in quadratic time. *J. Comb. Theory B*, 102(2):424–435, 2012. doi:10.1016/J.JCTB.2011.07.004.
- 16 Tuukka Korhonen, Michał Pilipczuk, and Giannos Stamoulis. Minor containment and disjoint paths in almost-linear time. *arXiv preprint arXiv:2404.03958*, 2024.
- 17 Dániel Marx, Marcin Pilipczuk, and Michał Pilipczuk. On subexponential parameterized algorithms for Steiner tree and directed subset TSP on planar graphs. In *Proceedings of the 59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018*, pages 474–484. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00052.

- 18 Kazuhiko Matsumoto, Takao Nishizeki, and Nobuji Saito. An efficient algorithm for finding multicommodity flows in planar networks. *SIAM J. Comput.*, 14(2):289–302, 1985. doi:10.1137/0214023.
- 19 Marcin Pilipczuk, Michal Pilipczuk, Piotr Sankowski, and Erik Jan van Leeuwen. Network sparsification for Steiner problems on planar and bounded-genus graphs. *ACM Trans. Algorithms*, 14(4):53:1–53:73, 2018. doi:10.1145/3239560.
- 20 J. Scott Provan. An approximation scheme for finding Steiner trees with obstacles. *SIAM J. Comput.*, 17(5):920–934, 1988. doi:10.1137/0217057.
- 21 Alexander Schrijver et al. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.
- 22 Paul Wollan. Extremal functions for rooted minors. *Journal of Graph Theory*, 58(2):159–178, 2008. doi:10.1002/jgt.20301.
- 23 David R. Wood and Svante Linusson. Thomassen’s choosability argument revisited. *SIAM Journal on Discrete Mathematics*, 24(4):1632–1637, 2010. doi:10.1137/100796649.

Kick the Cliques

Gaétan Berthe 


LIRMM, Université de Montpellier, CNRS, Montpellier, France

Marin Bougeret 

LIRMM, Université de Montpellier, CNRS, Montpellier, France

Daniel Gonçalves 

LIRMM, Université de Montpellier, CNRS, Montpellier, France

Jean-Florent Raymond 

Univ. Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, Lyon, France

Abstract

In the K_r -HITTING problem, given a graph G and an integer k one has to decide if there exists a set of at most k vertices whose removal destroys all r -cliques of G .

In this paper we give an algorithm for K_r -HITTING that runs in subexponential FPT time on graph classes satisfying two simple conditions related to cliques and treewidth. As an application we show that our algorithm solves K_r -HITTING in time

- $2^{O_r(k^{(r+1)/(r+2)} \log k)} \cdot n^{O_r(1)}$ in pseudo-disk graphs and map-graphs;
- $2^{O_{t,r}(k^{2/3} \log k)} \cdot n^{O_r(1)}$ in $K_{t,t}$ -subgraph-free string graphs; and
- $2^{O_{H,r}(k^{2/3} \log k)} \cdot n^{O_r(1)}$ in H -minor-free graphs.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Fixed parameter tractability; Theory of computation → Computational geometry

Keywords and phrases Subexponential FPT algorithms, implicit hitting set problems, geometric intersection graphs

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.13

Related Version *arXiv Version*: <https://arxiv.org/abs/2407.01465>

Funding *Jean-Florent Raymond*: Supported by the ANR project GRALMECO (ANR-21-CE48-0004).

1 Introduction

In the K_r -HITTING problem, given a graph G and an integer k one has to decide if there are k vertices in G whose deletion yields a K_r -free graph. This problem falls within the general family of (implicit) hitting problems and encompasses several extensively studied problems such as the case $r = 2$ better known under the name VERTEX COVER and the case $r = 3$ that we usually refer to as TRIANGLE HITTING. Already for these small values the problem is NP-complete.

In this paper we are interested in *subexponential parameterized algorithms* for K_r -HITTING, i.e., algorithms that run in *Fixed Parameter Tractable (FPT)* time (that is, time $f(k) \cdot n^{O(1)}$ for some computable function f) and where additionally the contribution of the parameter k is subexponential, in other words $f(k) \in 2^{o(k)}$. Under the Exponential Time Hypothesis of Impagliazzo and Paturi [10], such algorithms do not exist in general for vertex deletion problem to nontrivial hereditary properties [13] (like K_r -HITTING problem) and so we have to focus on particular graph classes.

Historically, subexponential graph algorithms were first obtained for specific problems in sparse graph classes such as planar graphs. The techniques used have then been unified and extended by Demaine, Fomin, Hajiaghayi, and Thilikos in the meta-algorithmic theory of



© Gaétan Berthe, Marin Bougeret, Daniel Gonçalves, and Jean-Florent Raymond; licensed under Creative Commons License CC-BY 4.0

19th International Symposium on Parameterized and Exact Computation (IPEC 2024).

Editors: Édouard Bonnet and Paweł Rzażewski; Article No. 13; pp. 13:1–13:17



Leibniz International Proceedings in Informatics

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

bidimensionality [7], which provides a generic machinery to solve a wide range of problems in subexponential FPT time on H -minor free graphs. Initially bidimensionality was defined for graph classes with some “flatness” property similar to planar graphs, typically graphs of bounded genus and H -minor-free graphs. Over the years, the theory saw several improvements and extensions in order to deal with different settings like map graphs and other classes of intersection graphs, which are initially not sparse as they contain large cliques for example, but where we can branch in subexponential-time to reduce to sparse instances (see for instance the bibliography cited in [4]). However, despite its generality, bidimensionality can only handle the so-called *bidimensional problems* where, informally, as soon as the instance (G, k) contains a large $t \times t$ grid as a minor (for $t \in o(k)$, typically $t = \sqrt{k}$), we know that (G, k) is necessarily a no-instance (or yes-instance depending on the problem). This is the case of VERTEX COVER but unfortunately not of TRIANGLE HITTING (as grids are triangle-free) and more generally not of K_r -HITTING for $r \geq 3$.

The focus of this paper is on this blind spot: subexponential FPT algorithms for a problem that is not bidimensional, namely K_r -HITTING. About this problem, we note that using arguments developed in the context of approximation [9], the following subexponential FPT algorithm can be obtained for apex-minor free graphs (which are sparse).

► **Theorem 1** (from [9]). *For every apex¹ graph H and every $r \in \mathbb{N}$ there is an algorithm solving K_r -HITTING in H -minor-free graphs in time $2^{O_{H,r}(\sqrt{k})} \cdot n^{O_r(1)}$.*

Regarding classes that are not sparse, TRIANGLE HITTING received significant attention in the last years in classes of intersection graphs such as (unit) disk graphs², pseudo-disk graphs, and subclasses of segment graphs³ [14, 2, 4, 3]. We only recall below the most general results and do not mention those that require a geometric representation.

► **Theorem 2.** *There are algorithms that given a parameter k and a n -vertex graph (without a geometric representation) solve TRIANGLE HITTING in time*

1. $2^{O(k^{3/4} \log k)} n^{O(1)}$ in disk graphs [2];⁴
2. $2^{O(k^{3/4} \log k)} n^{O(1)}$ in contact-segment⁵ graphs [4];
3. $2^{O_{t,d}(k^{2/3}) \log k} n^{O(1)}$ in $K_{t,t}$ -subgraph-free d -DIR⁶ graphs [4].

► **Theorem 3** ([4]). *Assuming the Exponential Time Hypothesis, there is no algorithm solving TRIANGLE HITTING in time*

1. $2^{o(n)}$ in 2-DIR graphs;
2. $2^{o(\sqrt{\Delta n})}$ in 2-DIR graphs with maximum degree Δ ; and
3. $2^{o(\sqrt{n})}$ in $K_{2,2}$ -free contact-2-DIR graphs of maximum degree 6.

Our contribution

Our main result is the following subexponential parameterized algorithm for K_r -HITTING in graph classes satisfying two conditions related to cliques and treewidth. Notice that the statement of the following theorem is a simplified version of the actual Theorem 22 that we prove in Section 5.

¹ A graph is *apex* if the deletion of some vertex yields a planar graph.

² (*Unit*) *disk graphs* are intersection graphs of (unit) disks in \mathbb{R}^2 .

³ *Segment graphs* are intersection graphs of segments in \mathbb{R}^2 .

⁴ The published version of the paper gives a bound of $2^{O(k^{4/5} \log k)} n^{O(1)}$ but it can easily be improved to $2^{O(k^{3/4} \log k)} n^{O(1)}$, as confirmed to us by the authors of [2] (private communication).

⁵ *Contact-segment graphs* are the intersection graphs of non-crossing segments in \mathbb{R}^2 .

⁶ A graph is d -DIR if it is the intersection graph of segments of \mathbb{R}^2 with at most d different slopes.

► **Theorem 4.** *Let $r \in \mathbb{N}$, $\alpha \in (0, 1)$, $\mu \in \mathbb{R}_{>0}$ and let \mathcal{G} be a hereditary graph class where every $G \in \mathcal{G}$ with n vertices and clique number ω has $O_r(\omega^\mu n)$ cliques of order less than r and treewidth $O_r(\omega^\mu n^\alpha)$. There exists $\varepsilon < 1$ and an algorithm that solves K_r -HITTING on \mathcal{G} in time $2^{k^\varepsilon} \cdot n^{O_r(1)}$.*

One additional motivation for this work was to generalize to K_r -HITTING the techniques used in previous work to solve TRIANGLE HITTING (in specific graph classes) and to extract the minimal requirements for such an approach to work in more general settings. We believe we met this goal as we actually describe a single generic approach that solves K_r -HITTING on any input graph, for any r . The properties of the class in which the inputs are taken is only used to bound its running time. Such a generalization effort can be fruitful and indeed it allowed us afterwards to identify natural graph classes where subexponential algorithms exist as a consequence of our general result, as we detail now.

In Section 6 we derive from Theorem 22 the following applications.

► **Theorem 5.** *There is an algorithm solving K_r -HITTING in pseudo-disk graphs in time $2^{O_r(k^{(r+1)/(r+2)} \log k)} \cdot n^{O_r(1)}$.*

Pseudo-disk graphs are a classical generalization of disk graphs where to each vertex is associated a *pseudo-disk* (a subset of the plane that is homeomorphic to a disk), two vertices are adjacent if the corresponding pseudo-disks intersect and additionally we require that for any two intersecting pseudo-disks, their boundaries intersect on at most two points. Disk graphs and contact segment graphs are pseudo-disk graphs, so Theorem 5 applies to the two settings handled by the algorithms of [2] and [4] mentioned at items 1 and 2 of Theorem 2. Another application is the following:

► **Theorem 6.** *There is an algorithm solving K_r -HITTING in map graphs⁷ in time $2^{O_r(k^{(r+1)/(r+2)} \log k)} \cdot n^{O_r(1)}$.*

We cannot expect a similar consequence for the more general class of string graphs.⁸ Indeed, there are n -vertex string graphs that are triangle-free and have treewidth $\Omega(n)$, for instance the balanced bicliques.⁹ Note that such graphs prevent string graphs from satisfying the requirement of Theorem 4. Also, and more importantly, by Theorem 3 under ETH there is no $2^{o(n)}$ -time algorithm for K_3 -HITTING in 2-DIR graphs, a restricted subclass of string graphs. As we will show, large bicliques are the only obstructions in the sense that forbidding them in string graphs allows us to solve the problem in subexponential FPT time. For this we use the following light version of Theorem 4 (also consequence of Theorem 22) suited for classes where the clique number is already bounded.

► **Theorem 7.** *Let $r \in \mathbb{N}$, $\alpha \in (0, 1)$ and let \mathcal{G} be a hereditary graph class where every $G \in \mathcal{G}$ with n vertices has $O(n)$ cliques of order less than r and treewidth $O(n^\alpha)$. There exists an algorithm that solves K_r -HITTING on \mathcal{G} in time $2^{O_r(k^{2/(1+1/\alpha)} \log k)} \cdot n^{O_r(1)}$.*

As a consequence we obtain a subexponential FPT algorithm for string graphs excluding large bicliques.

⁷ *Map graphs* are intersection graphs of interior-disjoint regions of \mathbb{R}^2 homeomorphic to disks.

⁸ *String graphs* are intersection graphs of Jordan arcs in \mathbb{R}^2 . They generalize many of the most studied classes of intersection graphs of geometric objects in the plane such as disk graphs, pseudo-disk graphs, segment graphs, chordal graphs, etc.

⁹ $K_{n,n}$ can be drawn as a 2-DIR graph with n horizontal disjoint segments that are all crossed by n vertical disjoint segments.

13:4 Kick the Cliques

► **Theorem 8.** *There is an algorithm solving K_r -HITTING in $K_{t,t}$ -subgraph-free string graphs in time $2^{O_{t,r}(k^{2/3} \log k)} \cdot n^{O_r(1)}$.*

Theorem 8 is a generalization in two directions (the objects to hit and the graph to consider) of item 3 of Theorem 2. We note that under ETH the contribution of k cannot be improved to $2^{o(\sqrt{k})}$, according to Theorem 3. Finally we observe that Theorem 4 can also be applied to certain classes of sparse graphs.

► **Theorem 9.** *For every graph H , there is an algorithm solving K_r -HITTING in H -minor-free graphs in time $2^{O_{H,r}(k^{2/3} \log k)} \cdot n^{O_r(1)}$.*

This is a more general statement than Theorem 1 in the sense that we are not limited to apex-minor free graphs, with the price of a slightly larger time complexity.

Our techniques

Our subexponential algorithm for K_r -HITTING of Theorem 4 is obtained as follows. Given (G, k) , we first perform in Section 3 a preliminary branching step whose objective is to get rid of large cliques (i.e., cliques of order at least k^ε for some $\varepsilon \in (0, 1)$ that we will fix later). This step is a folklore technique which is frequently used for any problem where a solution has to contain almost all vertices of a large clique, like TRIANGLE HITTING (or in general K_r -HITTING), FEEDBACK VERTEX SET, or ODD CYCLE TRANSVERSAL. After this preprocessing has been performed we can assume that the instances to solve have no clique on more than k^ε vertices. Then, we greedily compute an r -approximate K_r -hitting set M . If $|M| > kr$ we can already answer negatively, so in the following we may assume $|M| \leq kr$ and will use the fact that there is no K_r in $G - M$.

Now, the crucial part of the algorithm is Section 4. Informally, the goal of the algorithms described in this section is to extend M into a superset M' together with a new parameter $k' \leq k$, such that $|M'| = O(k^{1+c\varepsilon})$ (for some $c > 0$) and that vertices of $V(G) \setminus M'$ are irrelevant for the problem of hitting r -cliques. In that way, we can remove them, and it remains only to solve $(G[M'], k')$, whose treewidth can be typically bounded by $\sqrt{\omega(G)|M'|}$ in the graph classes we consider. As $\sqrt{\omega(G)|M'|} = O(k^{1/2 + \frac{c+1}{2}\varepsilon})$, this leads to a subexponential algorithm. We stress that the high-level description above is not a kernelization because first we actually do not produce a single reduced instance but instead we have to branch and obtain a subexponential (in k) number of sub-instances and second because the reduction steps are not carried out in polynomial time, but in subexponential (in k) time.

To obtain this set M' , we use lemmatas 14 and 21 that are inspired from the following “virtual branching” procedure of [14, Lemma 6.5]. This routine was introduced for TRIANGLE HITTING and works as follows. It starts with a triangle hitting set M (obtained by greedily packing disjoint triangles), and outputs a slightly larger superset M' such that vertices in $G - M'$ are *almost* useless, in the sense that every triangle has at least two vertices in M' (we do not detail here how to handle the triangles with exactly two vertices in M' and refer to [14]). This is done as follows. For a vertex $v \in M$, consider a maximum matching $M(v) \subseteq N(v) \cap (V(G) \setminus M)$. If for every $v \in M$ such a matching is small, meaning $|M(v)| \leq k^\varepsilon$, then we can define $M' = M \cup \bigcup_{v \in M} M(v)$. We are done as $|M'| = O(k^{1+\varepsilon})$ remains small, and there is no longer a $v \in M'$ with an edge in $N(v) \setminus M'$ (as this would form a triangle outside M). Otherwise, if for some $v \in M$, $|M(v)| > k^\varepsilon$, a solution of TRIANGLE HITTING should either take v , or otherwise hits all edges of $M(v)$. In the second case, it would be too costly to guess which vertex is taken in each $e \in M(v)$, so instead the procedure “absorbs” $M(v)$ by defining $M' = M \cup M(v)$. This absorption increases the size of M , but

“virtually” decreases the parameter k , as it increases by $|M(v)|$ the size of a matching that the solution will have to hit. This leads to a running time typically dominated by the recurrence $f(k) = f(k-1) + f(k-k^\varepsilon)$, which is subexponential in k .

Now, coming back to K_r -HITTING, given a set M , let us say¹⁰ that a *type- i clique* is an r -clique X such that $|X \cap M| = i$. We could remove type-1 cliques by using the previous virtual branching procedure, defining now $M(v)$ as a packing of $r-1$ cliques instead of a packing of edges, but the problem is that, if we want to obtain a set M' as promised (where vertices of $V(G) \setminus M'$ are useless), we also have to remove type- i cliques for $i \in \{2, \dots, r-1\}$. However, there is a first obstacle to remove such type- i cliques: as the part common with M (which was before a single vertex in M) is now an i -clique, we cannot afford to enumerate all possible choices X' of such an i -clique in M . Indeed, already for $i = 2$ we would possibly consider a quadratic number of sets X' , so absorbing every packing (of $(r-i)$ -cliques) $M(X')$ (in the unfortunate case where these are all small) would result in a set $M' = M \cup \bigcup_{X' \subseteq M, |X'|=2} M(X')$ with $|M'| = \Omega(k^2)$, which is too large for our purpose. To circumvent this issue, we identified a key property that holds in many geometric graph classes like pseudo-disk or $K_{t,t}$ -subgraph-free string graphs: in such graphs, there is only a linear (in the number of vertices) number of i -cliques for fixed i , and for fixed clique number ω . In our case, as ω is small, and $i \leq r$ is fixed, this implies that there are $O(|M|)$ such i -cliques in M . Hence, it allows us to control the size of M' . Of course, dealing with r -cliques instead of triangles also raises other problems, in particular related to the way we hit their intersection with M which is no longer a single vertex but a clique. To deal with this issue we had to introduce an annotated variant of the problem where additional sets of vertices have to be hit besides r -cliques.

Organization of the paper

In Section 2 we give the necessary definitions. We describe the first branching in Section 3 and the second in Section 4. The algorithm is given in Section 5. We give applications to selected graph classes in Section 6. We conclude with open questions in Section 7.

2 Preliminaries

Running times

When stating results related to algorithms, the variable n in the running time always refers to the number of vertices of the graph that is part of the input. For any parameter p (typically a graph H , or an integer r) and integer k , and any functions $f(p, k)$ and $g(k)$, we write $f = O_p(g(k))$ to indicate that for any fixed p , the restricted function $k \rightarrow f(p, k)$ is $O(g(k))$. To simplify the presentation we will assume that r is a fixed constant instead of explicitly give it as a parameter in all our algorithms and lemmas.

Graphs

Unless otherwise stated we use standard graph theory terminology. A *clique* in a graph G is a complete subgraph and when there is no ambiguity we also use *clique* to denote a subset of $V(G)$ inducing a complete subgraph. The *clique number* of G is the maximum number of vertices of a clique it contains and we denote it by $\omega(G)$. For any $i \in \mathbb{N}$, an *i -clique* is a

¹⁰This notion of type- i clique will not be used later and is just introduced for this sketch.

13:6 Kick the Cliques

clique on i vertices and a $(< i)$ -clique is a clique on less than i vertices. For any graph G and subset of vertices $X \subseteq V(G)$, we denote $G - X$ the graph whose vertex set is $V(G) \setminus X$ and edge set is $\{e \in E(G) : e \cap X = \emptyset\}$. Let H be a graph. We say that G is H -free if G does not contain H as induced subgraph.

Hypergraphs and hitting sets

A *hypergraph* is simply a collection of sets, where any set is referred as an *hyperedge*. So $|\mathcal{D}|$ refers to the number of sets in the hypergraph \mathcal{D} and we define $V(\mathcal{D}) = \bigcup_{D \in \mathcal{D}} D$. By $\mathcal{K}_r(G)$ (resp. $\mathcal{K}_{<r}(G)$) we denote the hypergraph of r -cliques (resp. $(< r)$ -cliques) of G , i.e. $\mathcal{K}_r(G) = \{X \subseteq V(G), X \text{ is an } r\text{-clique}\}$.

A *hitting set* of \mathcal{D} is a subset $X \subseteq V(\mathcal{D})$ that intersects every hyperedge of \mathcal{D} . A *matching* of \mathcal{D} is a collection of disjoint hyperedges. The maximum size of a matching in \mathcal{D} is denoted by $\nu(\mathcal{D})$. Note that a hitting set of \mathcal{D} has always size at least $\nu(\mathcal{D})$ as it needs to intersect each of the elements of a maximum matching, which are disjoint.

Special cases of hitting sets of hypergraphs are the hitting sets of subgraphs of a graph. For G, H two graphs, an H -hitting set in G is a subset $X \subseteq V(G)$ such that $G - X$ is H -free. In other words it is a hitting set of the hypergraph of the (induced) subgraphs of G isomorphic to H . In the H -HITTING problem, given a graph G and an integer $k \in \mathbb{N}$, one has to decide whether G has a H -hitting set of size at most k .

3 Dealing with large cliques

A K_r -HITTING can be useful to detect large cliques, as we explain now. This will allow us to identify cliques on which to branch.

► **Lemma 10.** *Given a graph G , a K_r -hitting set M , and an integer $p > r$, one can find a p -clique of G , or correctly conclude none exists, in $O(p^2|M|p n^{r-1})$ steps.*

Proof. For each choice of $p - r + 1$ vertices of M and $r - 1$ other vertices of G we check whether they form a clique (which takes $O(p^2)$ time), in which case return it and stop. If no clique is found we return that G is K_p -free. The correctness follows from the following observation: as $G - M$ is K_r -free, every p -clique of G has at least $p - r + 1$ of its vertices in M . ◀

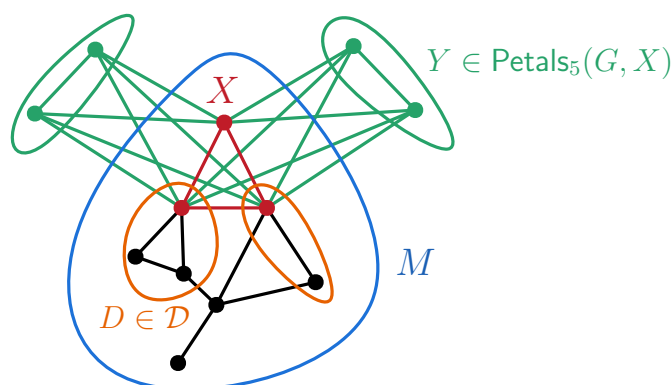
As noted in the proof of Lemma 10, every large clique of the input graph will have most of its vertices in a K_r -hitting set so we can branch on which these are.

► **Lemma 11.** *There is an algorithm that, given two integers $r \geq 1$ and $p > r$, an instance (G, k) of K_r -HITTING and a K_r -hitting set M of G , runs in $2^{kr(\log p)/p} |M|^p n^{O(r)}$ steps and returns a collection \mathcal{Y} of $2^{kr(\log p)/p} \cdot |M|^p$ instances of the same problem such that:*

1. (G, k) is a yes-instance if and only if \mathcal{Y} contains a yes-instance;
2. for every $(G', k') \in \mathcal{Y}$, G' has no p -clique.

Proof. The algorithm is the following:

1. Using the algorithm of Lemma 10 on G and M , we find a p -clique K (if none is found return $\mathcal{Y} = \{(G, k)\}$).
2. We initialize $\mathcal{Y} = \emptyset$.
3. Observe that any solution contains at least $p - r + 1$ vertices from K . For every subset X of K with $p - r + 1 \leq |X| \leq k$ vertices:



■ **Figure 1** A 3-clique X with two 5-petals that live in $G - M$. Here \mathcal{D} contains two hyperedges, represented in orange. Observe that no hyperedge of \mathcal{D} is contained in X , so X is a lush 3-clique.

- a. We look for solutions S that contain X with a recursive call on $(G - X, k - |X|)$ and $M \setminus X$;
 - b. We then add the resulting collection of instances to \mathcal{Y} .
4. Return \mathcal{Y} .

Regarding correctness, Item (2) follows from the base case of the recursion (step 1) and Item (1) can be proved by induction on the number of recursive calls, using the observation that (G, k) has a solution containing a set X if and only if $(G - X, k - |X|)$ has a solution.

We denote by $T_{r,p}(n, k)$ the time complexity of the above algorithm with parameters $r, p, (G, k)$ where $|G| = n$. The time taken by each computation step is the following:

1. Finding a p -clique takes time $O(p^2|M|^pn^{r-1})$ by Lemma 10.
2. When a p -clique K is found we consider at most p^{r-1} subsets X on which we perform a recursive call of cost at most $T_{r,p}(n - p + r - 1, k - p + r - 1)$.

So

$$T_{r,p}(n, k) \in |M|^pn^{O(r)} + p^{r-1} \cdot T_{r,p}(n - p + r - 1, k - p + r - 1).$$

We deduce

$$\begin{aligned} T_{r,p}(n, k) &\in p^{k(r-1)/(p-r+1)}|M|^pn^{O(r)} \\ &\in 2^{kr(\log p)/p}|M|^pn^{O(r)}, \end{aligned}$$

as desired. Note that a similar recurrence can be used to bound the number of output instances. ◀

4 Picking petals

Let $i \in \{1, \dots, r - 1\}$ and let X be an i -clique in a graph G . An r -petal of X is a subset of vertices of $G - X$ that together with X forms an r -clique. We denote by $\text{Petals}_r(G, X)$ the hypergraph of r -petals of X , i.e., $\text{Petals}_r(G, X) = \{Y \subseteq V(G) \setminus X, X \cup Y \in \mathcal{K}_r(G)\}$. See Figure 1 for an illustration.

In order to deal more easily with the recursive steps in our algorithms we introduce ANN- K_r -HITTING, an annotated version of K_r -HITTING where a number of choices have already been made, which is recorded by extra vertex subsets that the solution is required to hit.

In the ANN- K_r -HITTING problem, one is given a triple (G, \mathcal{D}, k) where G is a graph, $\mathcal{D} \subseteq \mathcal{K}_{<r}(G)$ and k is an integer. A *solution* to this instance is a set of vertices that hits $\mathcal{K}_r(G)$ and \mathcal{D} and has at most k vertices. The question is whether the input instance admits a solution.

In the forthcoming algorithms it will also be more convenient to consider, together with an instance, a non-optimal solution M . This motivates the following definition. A *context* is a pair $((G, \mathcal{D}, k), M)$ where (G, \mathcal{D}, k) is an instance of ANN- K_r -HITTING, M is a K_r -hitting set (possibly larger than k), and $V(\mathcal{D}) \subseteq M$. We say that a context is *positive* if the instance of ANN- K_r -HITTING it contains is a yes-instance, and *negative* otherwise.

Given a context $((G, \mathcal{D}, k), M)$ and $i \in \{1, \dots, r-1\}$, a *lush i -clique* is an i -clique X of $G[M]$ that has an r -petal in $G - M$ and such that no $D \in \mathcal{D}$ is subset of $V(X)$. See Figure 1 for an example.

Informally, if X is a lush clique then we are not guaranteed that hitting \mathcal{D} alone does always also hit the cliques induced by X and its petals, so we have to take care of them separately. Ideally we would like to get rid of lush cliques so that we can focus on \mathcal{D} to solve the problem. We say that the context $((G, \mathcal{D}, k), M)$ is *i -stripped* if for every $i' < i$ it has no lush i' -clique. These notions are motivated by the following easy lemma.

► **Lemma 12.** *Let $((G, \mathcal{D}, k), M)$ be an r -stripped context. The instances (G, \mathcal{D}, k) and $(G[M], \mathcal{D}, k)$ of ANN- K_r -HITTING are equivalent.*

Proof. First, recall that as $((G, \mathcal{D}, k), M)$ is a context, $V(\mathcal{D}) \subseteq M$ so $(G[M], \mathcal{D}, k)$ is indeed a valid instance of ANN- K_r -HITTING. Also as $\mathcal{K}_r(G[M]) \subseteq \mathcal{K}_r(G)$, if (G, \mathcal{D}, k) has a solution then $(G[M], \mathcal{D}, k)$ does. So we only need to show the other direction. Let $v \in V(G) \setminus M$ and suppose that G contains some r -clique X with $v \in X$. The set $M \cap X$ is not empty because M is a K_r -hitting set. Note that $X \cap M$ has a petal $X \setminus M$ disjoint from M . It cannot form a lush i -clique (for $i = |M \cap X| < r$) as this would contradict the assumption that the considered context is r -stripped, so there is some $D \in \mathcal{D}$ that is subset of $X \cap M$. Therefore every solution of $(G[M], \mathcal{D}, k)$ does hit X in G , as it hits D . As this holds for every v and X as above, every solution of $(G[M], \mathcal{D}, k)$ is a solution of (G, \mathcal{D}, k) , as desired. ◀

By the above lemma, if we manage to get rid of lush cliques, we can obtain an equivalent instance whose graph is not larger than $G[M]$. As we will see, we are able to handle lush cliques with the price of slightly increasing the size of M and producing several instances to represent the solutions of the original instance.

► **Lemma 13.** *There is an algorithm that, given an integer $i < r$ and an i -stripped context $((G, \mathcal{D}, k), M)$, runs in time $n^{O(r)}$ and either correctly concludes that the context is $(i+1)$ -stripped, or returns a lush i -clique X .*

Proof. The input context is i -stripped so we only have to check whether it contains a lush i -clique. We iterate over the i -cliques of $G[M]$. For every such clique X , we first check if $D \subseteq V(X)$ for some $D \in \mathcal{D}$. If so X is not a lush i -clique so we can move to the next choice of X . Otherwise we check if the common neighborhood of the vertices of X in $G - M$ has an $(r-i)$ -clique. If so this is an r -petal so X is lush and we can return it. Otherwise we continue to the next choice of X . If the iteration terminates without detecting a lush i -clique, we can safely return that the input context is $(i+1)$ -stripped. The complexity bound follows from the fact that $|\mathcal{D}| = n^{O(r)}$ and that $G[M]$ has $n^{O(r)}$ i -cliques. ◀

The following lemma is the key branching step in our subexponential algorithms for K_r -HITTING.

► **Lemma 14.** *There is an algorithm that, given $i \in \{1, \dots, r-1\}$ and $\lambda \in \{1, \dots, k\}$ and an i -stripped context $((G, \mathcal{D}, k), M)$ where we denote by ζ the number of i -cliques in $G[M]$, runs in time $2^{O((k/\lambda) \cdot \log \zeta)} \cdot n^{O(r)}$ and returns a collection \mathcal{Z} of size $2^{O((k/\lambda) \cdot \log \zeta)}$ of $(i+1)$ -stripped contexts such that:*

1. $((G, \mathcal{D}, k), M)$ is a positive context if and only if \mathcal{Z} contains one; and
2. for every $((G', \mathcal{D}', k'), M') \in \mathcal{Z}$, $M \subseteq M'$, $|M'| \leq |M| + r(\lambda\zeta + k)$, and $k' \leq k$.

Proof. Let us define an auxiliary algorithm that takes as input $(i, \lambda, (G, \mathcal{D}, k), M, \mathcal{P}^*)$, where $(i, \lambda, (G, \mathcal{D}, k), M)$ is as specified in the statement of the lemma, and \mathcal{P}^* is a matching of \mathcal{D} , and output the promised collection \mathcal{Z} . Such an auxiliary algorithm will imply the lemma, as we will run it with parameters $(i, \lambda, (G, \mathcal{D}, k), M, \emptyset)$. For simplicity, as i and λ will not change between recursive calls: we denote by $((G, \mathcal{D}, k), M, \mathcal{P}^*)$ the parameter of this auxiliary algorithm, which is defined as follows:

1. If $k < |\mathcal{P}^*|$, we can immediately return $\mathcal{Z} = \emptyset$.
2. Run the algorithm of Lemma 13 on i and $((G, \mathcal{D}, k), M)$. If no lush i -clique is found then the input context is already $(i+1)$ -stripped so we return $\mathcal{Z} = \{((G, \mathcal{D}, k), M)\}$. Otherwise let X denote the lush i -clique we found.
3. Construct the hypergraph \mathcal{B}_X of those r -petals of X that are subset of $V(G) \setminus M$.
4. Greedily compute a maximal matching \mathcal{P} in \mathcal{B}_X .
5. If $\tilde{\nu}_X \leq \lambda$, we return the result of the recursive call with parameters $((G, \mathcal{D}, k), M \cup V(\mathcal{P}), \mathcal{P}^*)$ (and quit).
6. Otherwise, we investigate the different ways to hit the r -cliques induced by X and \mathcal{B}_X (via X or via \mathcal{B}_X) as follows.
 - a. Solutions containing some (yet unspecified) vertex of X : this is done by a recursive call with parameters $((G, \mathcal{D} \cup \{X\}, k), M, \mathcal{P}^*)$. We call \mathcal{Z}_1 the resulting family.
 - b. Solutions hitting \mathcal{P} .¹¹ As \mathcal{P} is disjoint from M (hence from \mathcal{P}^*), such solutions exist only if $k \geq |\mathcal{P}^*| + |\mathcal{P}|$. In this case we define \mathcal{Z}_2 as the result of the recursive call with parameters

$$((G, \mathcal{D} \cup \mathcal{P}, k), M \cup V(\mathcal{P}), \mathcal{P}^* \cup \mathcal{P}).$$

Otherwise we set $\mathcal{Z}_2 = \emptyset$.

7. We return $\mathcal{Z}_1 \cup \mathcal{Z}_2$.

Observe first that in step (6b), the last parameter $\mathcal{P}^* \cup \mathcal{P}$ is a matching as required, as in particular $V(\mathcal{P}) \subseteq V(G) - M$ and $V(\mathcal{P}^*) \subseteq M$, by definition. We first prove the following fragment of item (2).

▷ **Claim 15.** For every $((G', \mathcal{D}', k'), M') \in \mathcal{Z}$, $M \subseteq M'$ and $k' \leq k$.

Proof. The only places where M is updated are steps 5 and 6b, where new vertices are added to it. Besides we never change the value of the parameter k . ◁

Let us describe the recursion tree T of the above algorithm on some input $((G, \mathcal{D}, k), M, \mathcal{P})$. The nodes of this tree are inputs. The root is $((G, \mathcal{D}, k), M, \emptyset)$ and a node s' is child of a node s if a call of the above algorithm on the input s triggers a call on the input s' . So the leaves of this tree are the inputs that do not trigger any recursive call.

¹¹ Notice that some of these solutions have possibly already been investigated in the previous step. For our purpose it is not an issue however to consider several times the same solution.

13:10 Kick the Cliques

Let us consider a path from the root of T to some leaf. We denote by

$$((G_j, \mathcal{D}_j, k_j), M_j, \mathcal{P}_j^*)_{j \in \{1, \dots, \ell\}}$$

the inputs along this path, and by $C_j = ((G_j, \mathcal{D}_j, k_j), M_j)$ the corresponding contexts, with $((G_1, \mathcal{D}_1, k_1), M_1, \mathcal{P}_1^*) = ((G, \mathcal{D}, k), M, \emptyset)$ and $((G_\ell, \mathcal{D}_\ell, k_\ell), M_\ell, \mathcal{P}_\ell^*)$ corresponding to the aforementioned leaf. Also, for every $j \in \{1, \dots, \ell - 1\}$ we denote by X_j the lush i -clique of C_j that is considered in the corresponding call.

We first show that all lush i -cliques considered along this path belong to the hitting set $M = M_1$ of the initial context.

▷ **Claim 16.** For every $j \in \{1, \dots, \ell - 1\}$, $X_j \subseteq M$.

Proof. Suppose towards a contradiction that for some j , $X_j \not\subseteq M$. Observe that since C_j is not a leaf, \mathcal{B}_{X_j} is not empty so X_j induces an r -clique together with some r -petal $B \in \mathcal{B}_{X_j}$. Recall that B is disjoint from M_j , by definition of \mathcal{B}_{X_j} . As M_j is a superset of M (Claim 15), B is disjoint from M as well. So M intersects X_j otherwise the r -clique $X_j \cup B$ would not be hit by M . Let $i' = |X_j \cap M|$. Note that $X_j \cap M$ is a lush i' -clique of C_1 since it has an r -petal $B \cup (X \setminus M)$ disjoint from M (the fact that no set of \mathcal{D} is subset of $X_j \cap M$ follows from the fact that this property holds for X_j). By our initial assumption and as $|X_j| = i$, we have $i' < i$. This contradicts the fact that C_1 is i -stripped. ◁

With a similar proof we can show the following (so the recursive calls are indeed made on valid inputs).

▷ **Claim 17.** For every $i \in \{1, \dots, \ell\}$, C_i is i -stripped.

Let us now show that each lush clique is only considered once.

▷ **Claim 18.** For every distinct $j, j' \in \{1, \dots, \ell - 1\}$, $X_j \neq X_{j'}$.

Proof. Suppose towards a contradiction that for some $j < j'$ we have $X_j = X_{j'}$. Then in the call on the context C_j , the next context C_{j+1} was obtained at step 5 or 6b (since at step 6a we would include X_j in \mathcal{D}_j , preventing it to be considered in future calls). In any of these two possibilities we set $M_{j+1} = M_j \cup V(\mathcal{P}_j)$, where \mathcal{P}_j denotes the maximal matching of step 4 in the call on context C_j .

Besides, as X_j is a lush i -clique in the context $C_{j'}$, then it has some r -petal B subset of $V(G_{j'}) - M_{j'}$. As $M_{j+1} \supseteq M_{j'}$ (Claim 15), B is also an r -petal of X_j and is subset of $V(G_j) - M_j$ and by the above observation, it is disjoint from \mathcal{P}_j . This contradicts the maximality of \mathcal{P}_j . ◁

Recall that the number of i -cliques in M is ζ . As a consequence of Claim 16 and Claim 18, we get the following.

▷ **Claim 19.** The recursion tree has depth at most ζ .

We can now conclude the proof of item (2).

▷ **Claim 20.** For every $((G', \mathcal{D}', k'), M') \in \mathcal{Z}$, $|M'| \leq |M| + r(\lambda\zeta + k)$.

Proof. When considering the context C_j , and given the chosen maximal matching \mathcal{P}_j of the petals of X_j , the set M_{j+1} is defined from M_j by:

- either adding the at most $\lambda(r - 1)$ new vertices of \mathcal{P}_j , if we make the recursive call at step 5,
- or by adding the vertices of the petals of \mathcal{P}_j , if we recurse at step 6b. Recall that in this case we also have $\mathcal{D}_{j+1} = \mathcal{D}_j \cup \mathcal{P}_j$ and $\mathcal{P}_{j+1}^* = \mathcal{P}_j^* \cup \mathcal{P}_j$

In the later case the number of added vertices is not directly bounded however we have $|\mathcal{P}_{j+1}^*| = |\mathcal{P}_j^*| + |\mathcal{P}_j|$. Because of the stopping condition of step 1, we will overall (from C_1 to C_ℓ) add at most k petals to the hypergraph and each petal has at most $r - 1$ vertices. Hence we get $|M'| \leq |M| + r(\lambda\zeta + k)$, as claimed. \triangleleft

Let us now show that the algorithm is correct, i.e., item 1 of the statement of the lemma. The proof is by induction on the depth of the recursion tree (i.e., $\ell - 1$ with the notation above). When the depth is 0, there is no recursive call. This corresponds to the two base cases in this algorithms: step 1, when the “budget” k is insufficient, and step 2, when the input context is already $(i + 1)$ -stripped. Clearly the outputs in these cases satisfy 1.

So we now consider the case of a run of the algorithm where the recursion tree has depth at least 1 and suppose that item 1 holds for all runs with recursion trees of smaller depth.

- If the recursive call is made at step 5 then item 1 trivially holds because the instance is unchanged.
- Otherwise, note that any solution has to hit the r -cliques induced by X and \mathcal{B}_X . So any solution either contains a vertex of X , or hits \mathcal{B}_X (or both). These are exactly the two branches that are explored in steps 6a and 6b, respectively, by our induction hypothesis.

The above shows that the algorithm is correct. It remains to prove that it has the claimed running time. Note that we do not update the graph neither the parameter between recursive calls: we will always work on the graph G with n vertices and with the parameter k . So the induction proving the time bound will use two different parameters as we explain now. For every $x, y \in \mathbb{N}$, let us denote by $T(x, y)$ the worst-case running time of the above algorithm on an input $((G, \mathcal{D}, k), M, \mathcal{P}^*)$ with $|G| = n$ such that there is at most x (non-necessarily disjoint) lush i -cliques in $G[M]$ and such that $|\mathcal{P}^*| \geq y$. Let $S_r(n)$ be the sum of the worst-case complexity of all subroutines needed in the different steps of item (1) to item (7) to the exception of recursive calls (one such subroutine is the algorithm of Lemma 13, another one is the construction of the hypergraph \mathcal{B}_X). Observe that $T(x, y) \leq S_r(n)$ when $y > k$ (as we fall into base case of step 1) or when $x = 0$ (as we fall into base case of step 2). Notice also that by definition we have $T(x, y) \leq T(x', y)$ for any $x \leq x'$, and $T(x, y) \leq T(x, y')$ for any $y' \leq y$.

If we make a recursive call at step 5, we return in time at most $T(x - 1, y)$ (by induction) as, by Claim 18, no lush i -clique of the original instance is considered twice. Otherwise, we will make recursive calls in step 6 which by induction take time at most $T(x - 1, y) + T(x - 1, y + |\mathcal{P}|)$ as, in the first branch 6a of recursion, Claim 18 implies again that no lush i -clique is considered two times, and in the second branch 6b, we know in addition that the size of the matching given as parameter increases by $|\mathcal{P}|$. Thus, in both cases (and including the other computation steps which take time $S_r(n)$), we obtain the upper bound:

$$\begin{aligned} T(x, y) &\leq T(x - 1, y) + T(x - 1, y + |\mathcal{P}|) + S_r(n) \\ &\leq T(x - 1, y) + T(x, y + \lambda) + S_r(n). \end{aligned}$$

(For the last line recall that T is anti-monotone with respect to its second parameter.)

Let us now show that for every $x, y \in \mathbb{N}$, $T(x, y) \leq xT(x, y + \lambda) + (x + 1)S_r(n)$. The proof is by induction on x . The base case $x = 0$ holds as observed above. Suppose the inequality holds for $x - 1$. As proved above

$$\begin{aligned} T(x, y) &\leq T(x - 1, y) + T(x, y + \lambda) + S_r(n) \\ &\leq (x - 1)T(x - 1, y + \lambda) + xS_r(n) && \text{(by induction)} \\ &\quad + T(x, y + \lambda) + S_r(n) \\ &\leq xT(x, y + \lambda) + (x + 1)S_r(n), && \text{as claimed.} \end{aligned} \tag{1}$$

13:12 Kick the Cliques

We now prove that for every $x, y \in \mathbb{N}$,

$$T(x, y) \leq x^{\frac{k+1-y}{\lambda}} + \left(1 + \frac{k+1-y}{\lambda}\right) (x+1)S_r(n).$$

This time the induction is on y . The base case $y > k$ hold as observed above. Let $x \geq 1$ and $y \leq k$ and suppose the inequality holds for any pair (x', y') with $y' > y$. Then as proved above in Eq. 1,

$$\begin{aligned} T(x, y) &\leq xT(x, y + \lambda) + (x+1)S_r(n) \\ &\leq x \cdot x^{\frac{k+1-y-\lambda}{\lambda}} + \left(1 + \frac{k+1-y-\lambda}{\lambda}\right) (x+1)S_r(n) + (x+1)S_r(n) \\ &\leq x^{\frac{k+1-y}{\lambda}} + \left(1 + \frac{k+1-y}{\lambda}\right) (x+1)S_r(n). \end{aligned}$$

Observe that $S_r(n)$ is dominated by the time spent in the algorithm of Lemma 13, and thus $S_r(n) \in n^{O(r)}$. As we initially have $x \leq \zeta$ and $y \geq 0$, we obtained the claimed running time. A similar analysis can be used to bound the size of the output family \mathcal{Z} . ◀

By iterating the algorithm of Lemma 14 for increasing values of i we can obtain a collection of r -stripped contexts, as we explain now.

► **Lemma 21 (picking petals).** *There is an algorithm that, given $i \in \{1, \dots, r\}$, $\lambda \in \mathbb{R}_{\geq 1}$, a context $((G, \mathcal{D}, k), M)$, where we denote by ζ the number of $(< r)$ -cliques in $G[M]$, runs in time $2^{O((i \cdot k/\lambda) \cdot \log \zeta)} \cdot n^{O(r)}$ and returns a collection \mathcal{Z} of size $2^{O((i \cdot k/\lambda) \cdot \log \zeta)}$ of i -stripped contexts such that:*

1. (G, \mathcal{D}, k) is a yes-instance if and only if some input of \mathcal{Z} contains one; and
2. for every $((G', \mathcal{D}', k'), M') \in \mathcal{Z}$, $M \subseteq M'$, $|M'| \leq |M| + ir(\lambda\zeta + k)$, and $k' \leq k$.

Proof. Again for the sake of clarity we assume λ is a fixed constant.

The proof is by induction on i . For the base case $i = 1$ we simply observe that $((G, \mathcal{D}, k), M)$ is already 1-stripped so there is nothing to do.

So let us now suppose that $i > 1$ and that the statement holds for $i - 1$. So from the input context $((G, \mathcal{D}, k), M)$ and $i - 1$ we can use the induction hypothesis to construct a collection \mathcal{Z}_{i-1} of $(i - 1)$ -stripped contexts satisfying the statement for $i - 1$. Now we apply the algorithm of Lemma 14 to $i - 1$ and each context in \mathcal{Z}_{i-1} and call \mathcal{Z}_i the union of the obtained collections. Item 1 follows from the properties of \mathcal{Z}_{i-1} and the correctness of the algorithm of Lemma 14. Constructing \mathcal{Z}_{i-1} takes time $2^{O(((i-1) \cdot k/\lambda) \cdot \log \zeta)} \cdot n^{O(r)}$ (by induction) and then we run the $(2^{O((k/\lambda) \cdot \log \zeta)} \cdot n^{O(r)})$ -time algorithm of Lemma 14 on each of its $2^{O(((i-1) \cdot k/\lambda) \cdot \log \zeta)}$ contexts. This results in an overall running time of $2^{O((i \cdot k/\lambda) \cdot \log \zeta)} \cdot n^{O(r)}$, as claimed. In each context of \mathcal{Z}_{i-1} the set M' has size at most $|M| + (i - 1)r(\lambda\zeta + k)$ (by induction) and after the run of the algorithm Lemma 14, the corresponding set in the produced instances has at most $r(\lambda\zeta + k)$ vertices more so we get the desired bound. Finally, as in the proof of Lemma 14 the value of k never changes. ◀

5 Kick the cliques

For every $\phi, \gamma \in \mathbb{R}_{\geq 0}$ and $\alpha \in (0, 1)$, we say that graph class \mathcal{G} has property $P_r(\phi, \gamma, \alpha)$ if there are functions $f(x) \in O(x^\phi)$ and $g(x) \in O(x^\gamma)$ such that for $G \in \mathcal{G}$ with n vertices and clique number less than ω ,

- (P1) G has at most $f(\omega) \cdot n$ cliques of order less than r ; and
(P2) $\text{tw}(G) \leq g(\omega)n^\alpha$.

Our main contribution is the following.

► **Theorem 22.** *For every hereditary graph class \mathcal{G} that has property $P_r(\phi, \gamma, \alpha)$ for some $\alpha \in (0, 1)$, $\phi, \gamma \in \mathbb{R}_{\geq 0}$, there is an algorithm that solves K_r -HITTING on \mathcal{G} in time*

$$2^{O_{r,\phi}(k^\varepsilon \log k)} \cdot n^{O_r(1)} \quad \text{with} \quad \varepsilon = \frac{\gamma + \alpha(\phi + 2)}{\gamma + \alpha(\phi + 1) + 1} < 1,$$

i.e., subexponential FPT time.

Proof. Let $f: x \mapsto c_f \cdot x^\phi$ and $g: x \mapsto c_g \cdot x^\gamma$ (for some $c_f, c_g > 0$) be as in the definition of property $P_r(\phi, \gamma, \alpha)$. For the sake of readability we will here use (when deemed useful) \exp to denote the function $x \mapsto 2^x$ defined over reals. Given an instance (G, k) of K_r -HITTING, we consider the context $((G, \mathcal{D}, k), M)$ where $\mathcal{D} = \emptyset$ and M is an r -approximation of a K_r -hitting set computed in $n^{O(r)}$ time by greedily packing disjoint r -cliques. If $|M| > kr$ we can already answer negatively, so in what follows we suppose that $|M| \leq kr$. We run the algorithm of Lemma 11 with $p = \lceil k^\varepsilon \rceil$ for some constant $\varepsilon \in (0, 1)$ that we will fix later. In time

$$2^{O(rk^{1-\varepsilon} \log k)} n^{O(r)}$$

we obtain a set \mathcal{Y} of $2^{O(rk^{1-\varepsilon} \log k)}$ contexts that have no p -clique. For each such context we apply the petal-picking algorithm of Lemma 21 with

$$\begin{aligned} \lambda &= k^\varepsilon & \text{and} \\ \zeta &= f(p) \cdot |M| \\ &\in O(rk^{1+\varepsilon\phi}). \end{aligned}$$

Let \mathcal{Z} denote the union of the outputs families of these algorithms. Computing this set then takes time

$$\begin{aligned} |\mathcal{Y}| \cdot 2^{O((r \cdot k/\lambda) \cdot \log \zeta)} \cdot n^{O(r)} &\in 2^{O(r \cdot k^{1-\varepsilon} (\log k + \log \zeta))} \cdot n^{O(r)} \\ &\in 2^{O(r^2 \cdot (1+\varepsilon\phi) k^{1-\varepsilon} \log k)} \cdot n^{O(r)}. \end{aligned}$$

For every $((G', \mathcal{D}', k), M') \in \mathcal{Z}$ we have

$$\begin{aligned} |M'| &\leq |M| + r^2(\lambda\zeta + k) & \text{by Lemma 21} \\ &\in O(kr + r^2(rk^{1+\varepsilon\phi+\varepsilon} + k)) \\ &\in O(r^3 k^{1+\varepsilon(\phi+1)}). \end{aligned}$$

So given any such context, we can decide whether it is positive or not in time

$$\begin{aligned} 2^{g(p)|M'|^\alpha} n^{O(1)} &\in \exp\left(O\left(p^\gamma \cdot \left(r^3 k^{1+\varepsilon(\phi+1)}\right)^\alpha\right)\right) \cdot n^{O_r(1)} \\ &\in \exp\left(O\left(r^{3\alpha} \cdot k^{\varepsilon\gamma + \alpha(1+\varepsilon(\phi+1))}\right)\right) \cdot n^{O_r(1)} \end{aligned}$$

as follows: first we use Lemma 12 to delete irrelevant vertices and obtain an equivalent instance H on $|M'|$ vertices, then we use property $P_r(\phi, \gamma, \alpha)$ to bound the treewidth of H and then we solve the problem by dynamic programming on an approximate tree-decomposition in $2^{O(\text{tw}(H))} n^{O(1)}$ time by noting that every r -clique and every $D' \in \mathcal{D}'$ (which is an $(< r)$ -clique) has to be contained in a bag.

13:14 Kick the Cliques

So overall, computing \mathcal{Z} and solving the problem in each sub-instance takes time

$$\begin{aligned} & \exp\left(O_r\left((1+\varepsilon\phi)k^{1-\varepsilon}\log k + k^{\varepsilon\gamma+\alpha(1+\varepsilon(\phi+1))}\right)\right) \cdot n^{O_r(1)} \\ & \in \exp\left(O_r\left((1+\varepsilon\phi)k^{1-\varepsilon}\log k + k^{\varepsilon(\gamma+\alpha+\alpha\phi)+\alpha}\right)\right) \cdot n^{O_r(1)}. \end{aligned}$$

As we aim for algorithms where the contribution of k to the time complexity is of the form $2^{o(k)}$, the above bound sets the following constraint: $\varepsilon < \frac{1-\alpha}{\gamma+\alpha+\alpha\phi}$. Let $\varepsilon = \frac{1-\alpha-\delta}{\gamma+\alpha+\alpha\phi}$ for some constant $\delta \in (0, 1)$ that we will fix later. Then the above complexity becomes:

$$\exp\left(O_r\left((1+\varepsilon\phi)k^{1-\varepsilon}\log k + k^{1-\delta}\right)\right) \cdot n^{O_r(1)}.$$

We optimize (ignoring logarithmic factors) by choosing the value of δ so that $1 - \varepsilon = 1 - \delta$, i.e., $\delta = \frac{1-\alpha}{\gamma+\alpha(\phi+1)+1}$. This gives the following overall time bound:

$$2^{O_r((1+\varepsilon\phi)k^{\varepsilon'}\log k)} \cdot n^{O_r(1)} \quad \text{with } \varepsilon' = \frac{\gamma + \alpha(\phi + 2)}{\gamma + \alpha(\phi + 1) + 1}.$$

As $\alpha < 1$ we have $\varepsilon' < 1$ so the algorithm runs in subexponential FPT time, as desired. ◀

6 Applications

In this section we give applications of Theorem 22 to specific graph classes. First we have to show that the considered classes satisfy property P_r (recall that this property is defined at the beginning of Section 5).

6.1 Pseudo-disk graphs and map graphs

In this subsection we will prove the following lemma.

► **Lemma 23.** *Pseudo-disk graphs and map graphs have the property $P_r(r-2, 1/2, 1/2)$.*

As a consequence we get the two following results.

► **Theorem 5.** *There is an algorithm solving K_r -HITTING in pseudo-disk graphs in time $2^{O_r(k^{(r+1)/(r+2)}\log k)} \cdot n^{O_r(1)}$.*

► **Theorem 6.** *There is an algorithm solving K_r -HITTING in map graphs¹² in time $2^{O_r(k^{(r+1)/(r+2)}\log k)} \cdot n^{O_r(1)}$.*

To prove Lemma 23, we first need to state some external results. For $d \in \mathbb{N}$ we say that a graph G is d -degenerate if every subgraph of G (including G itself) has a vertex of degree at most d . In order to bound the number of small cliques in the considered graphs we can bound their degeneracy and then rely on the following result of Chiba and Nishizeki.

► **Theorem 24 ([6]).** *Any string graph G with n vertices and degeneracy d has $O(id^{i-1}n)$ i -cliques.*

Actually [6] gives a time bound for the enumeration of i -cliques in graphs of arboricity d . As arboricity and degeneracy are linearly bounded by each other and since the time bound implies a bound on the number of enumerated objects (up to a constant factor), we get the above statement.

¹²Map graphs are intersection graphs of interior-disjoint regions of \mathbb{R}^2 homeomorphic to disks.

► **Theorem 25** ([3]). *Pseudo-disk graphs on n vertices with clique number ω have at most $3e\omega n$ edges and treewidth $O(\sqrt{\omega n})$. In particular they are $(3e\omega)$ -degenerate.*

► **Theorem 26** ([5]). *Map graphs on n vertices with clique number ω have at most $7\omega n$ edges. In particular they are (7ω) -degenerate.*

To bound the treewidth of map graphs we use the following combination of results on balanced separators of string graphs of Lee and the links between separators and treewidth of Dvořák and Norin.

► **Theorem 27** ([12] and [8]). *Any m -edge string graph has treewidth $O(\sqrt{m})$.*

As a consequence of Theorem 26 and Theorem 27 we get the following.

► **Corollary 28.** *Map graphs on n vertices with clique number ω have treewidth $O(\sqrt{\omega n})$.*

Proof of Lemma 23. Let G be a pseudo-disk graph with n vertices and clique number ω . By Theorem 25 the pseudo-disk graphs with n vertices and clique number ω are $(3e\omega)$ -degenerate, so by Theorem 24 they have $O((r-1)^2(3e\omega)^{r-2}n)$ cliques of order less than r . So property P1 holds with $\phi = r - 2$. By Theorem 25 property P2 is satisfied with $\alpha = 1/2$ and $\gamma = 1/2$. The proof for map graphs is very similar, using Theorem 26 and Corollary 28. ◀

6.2 String graphs

We now move to string graphs where, as discussed in the introduction, forbidding large bicliques is necessary. Actually for $K_{t,t}$ -subgraph free string graphs the branching of Lemma 10 to reduce the clique number is not necessary in the algorithm of Theorem 22 as the number of small cliques and the treewidth are already suitably bounded. This explains the zeroes in Lemma 29 hereafter. In this subsection we will prove the following lemma.

► **Lemma 29.** *$K_{t,t}$ -subgraph-free string graphs have the property $P_r(0, 0, 1/2)$.*

As a consequence we get the following result.

► **Corollary 30.** *There is an algorithm solving K_r -HITTING in $K_{t,t}$ -subgraph-free string graphs in time*

$$2^{O_{t,r}(k^{2/3} \log k)} \cdot n^{O_r(1)}.$$

The contributions of t and r to the complexity in Corollary 30 are not explicit due to the way we stated the bound of Theorem 22 but can be extracted from the proof of this theorem.

► **Theorem 31** ([12]). *For every $t \in \mathbb{N}$, $K_{t,t}$ -subgraph-free string graphs on n vertices have degeneracy $O(t \log t)$ and treewidth $O(\sqrt{n \cdot t \log t})$.*

Proof of Lemma 29. Combining Theorem 24 and Theorem 31 we get that the number of $(< r)$ -cliques in $K_{t,t}$ -subgraph-free string graphs is $O_r((t \log t)^{r-2}n)$, i.e. P1 holds with $\phi = 0$. Theorem 31 gives P2 with $\alpha = 1/2$ and $\gamma = 0$. ◀

6.3 Minor-closed classes

In this subsection we will prove the following lemma.

► **Lemma 32.** *For every graph H , H -minor-free graphs have the property $P_r(0, 0, 1/2)$.*

As a consequence we get the following result.

► **Theorem 9.** *For every graph H , there is an algorithm solving K_r -HITTING in H -minor-free graphs in time $2^{O_{H,r}(k^{2/3} \log k)} \cdot n^{O_r(1)}$.*

To prove Lemma 32, we first need to state some external results.

► **Theorem 33** (see [16]). *Every d -degenerate graph with $n \geq d$ vertices has at most $2^d(n - d + 1)$ cliques.*

► **Theorem 34** ([11, 15]). *For every h -vertex graph H there is a constant $d = O(h\sqrt{\log h})$ such that H -minor-free graphs are d -degenerate.*

► **Theorem 35** ([1]). *For every graph H , n -vertex H -minor-free graphs have treewidth $O_H(\sqrt{n})$.*

Proof of Lemma 32. As a consequence of Theorem 34 and Theorem 33, H -minor-free graphs have a linear number of cliques (regardless of their clique number). Hence they satisfy P1 with $\phi = 0$. By Theorem 35 they also satisfy P2 with $\alpha = 1/2$ and $\gamma = 0$. ◀

7 Open problems

As discussed in the introduction, our main result provides a generic way to obtain subexponential parameterized algorithms for K_r -HITTING, which can in particular be applied to several graph classes for which such algorithms were known from previous work (for TRIANGLE HITTING, the special case $r = 3$). Nevertheless there is still a gap between the running times of these different applications of our algorithm and the best time bounds for these specific classes. One can for instance compare our Theorem 5 (resp. Theorem 9) with the previous results corresponding to items 1 and 2 of Theorem 2 (resp. Theorem 1). It would be nice to match these known bounds, or to improve them when possible. More generally we can ask about the infimum ε such that K_r -HITTING can be solved in time $2^{O(k^\varepsilon)} n^{O(1)}$ in the classes we considered. We recall that under ETH, K_3 -HITTING cannot be solved in time $2^{o(\sqrt{n})}$ (so $\varepsilon \geq 1/2$) even for a very restricted subclass of string graphs (Theorem 3).

A second research direction is to understand for which graphs H our results can be extended to the H -HITTING problem (where one wants to hit any subgraph isomorphic to H). In disk graphs for example, it is already known [14] that there exist subexponential FPT algorithms for P_ℓ -HITTING when $\ell \leq 5$.

Recall that in this paper we gave sufficient conditions for a hereditary graph class to admit a subexponential FPT algorithm for K_r -HITTING. It remains an open problem to characterize such classes.

References


- 1 Noga Alon, Paul Seymour, and Robin Thomas. A separator theorem for nonplanar graphs. *Journal of the American Mathematical Society*, 3(4):801–808, 1990. URL: <http://www.jstor.org/stable/1990903>.
- 2 Shinwoo An, Kyungjin Cho, and Eunjin Oh. Faster algorithms for cycle hitting problems on disk graphs. In Pat Morin and Subhash Suri, editors, *Algorithms and Data Structures - 18th International Symposium, WADS 2023, Montreal, QC, Canada, July 31 - August 2, 2023, Proceedings*, volume 14079 of *Lecture Notes in Computer Science*, pages 29–42, Berlin, Heidelberg, 2023. Springer. doi:10.1007/978-3-031-38906-1_3.

- 3 Gaétan Berthe, Marin Bougeret, Daniel Gonçalves, and Jean-Florent Raymond. FVS for pseudo-disk graphs in subexponential FPT time. In *Proceedings of WG 2024*. LNCS, Springer, 2024.
- 4 Gaétan Berthe, Marin Bougeret, Daniel Gonçalves, and Jean-Florent Raymond. Subexponential Algorithms in Geometric Graphs via the Subquadratic Grid Minor Property: The Role of Local Radius. In Hans L. Bodlaender, editor, *19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024)*, volume 294 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:18, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SWAT.2024.11.
- 5 Zhi-Zhong Chen, Michelangelo Grigni, and Christos H Papadimitriou. Map graphs. *Journal of the ACM (JACM)*, 49(2):127–138, 2002. doi:10.1145/506147.506148.
- 6 Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on computing*, 14(1):210–223, 1985. doi:10.1137/0214017.
- 7 Erik D Demaine, Fedor V Fomin, Mohammadtaghi Hajiaghayi, and Dimitrios M Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and h-minor-free graphs. *Journal of the ACM (JACM)*, 52(6):866–893, 2005. doi:10.1145/1101821.1101823.
- 8 Zdeněk Dvořák and Sergey Norin. Treewidth of graphs with balanced separations. *J. Comb. Theory B*, 137:137–144, 2019. doi:10.1016/j.jctb.2018.12.007.
- 9 Fedor V Fomin, Daniel Lokshtanov, Venkatesh Raman, and Saket Saurabh. Bidimensionality and EPTAS. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 748–759. SIAM, 2011. doi:10.1137/1.9781611973082.59.
- 10 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.
- 11 A Kostochka. On the minimum of the Hadwiger number for graphs with given average degree. *Metody Diskret. Analiz.*, 38:37–58, 1982. English translation: AMS Translations (2), 132:15–32, 1986.
- 12 James R. Lee. Separators in Region Intersection Graphs. In Christos H. Papadimitriou, editor, *8th Innovations in Theoretical Computer Science Conference (ITCS 2017)*, volume 67 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1:1–1:8, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ITCS.2017.1.
- 13 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is np-complete. *J. Comput. Syst. Sci.*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4.
- 14 Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, Jie Xue, and Meirav Zehavi. Subexponential parameterized algorithms on disk graphs (extended abstract). In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2005–2031. SIAM, 2022. Full version available at <https://sites.cs.ucsb.edu/~daniello/papers/subexpDiskOCTandFriends.pdf>. doi:10.1137/1.9781611977073.80.
- 15 Andrew Thomason. An extremal function for contractions of graphs. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 95(2), pages 261–265. Cambridge University Press, 1984.
- 16 David R Wood. On the maximum number of cliques in a graph. *Graphs and Combinatorics*, 23(3):337–352, 2007. doi:10.1007/s00373-007-0738-8.

The Parameterized Complexity Landscape of Two-Sets Cut-Uncut

Matthias Bentert ✉

University of Bergen, Norway

Fedor V. Fomin ✉ 

University of Bergen, Norway

Fanny Hauser ✉

Technische Universität Berlin, Germany

University of Bergen, Norway

Saket Saurabh ✉ 

The Institute of Mathematical Sciences, Chennai, India

University of Bergen, Norway

Abstract

In TWO-SETS CUT-UNCUT, we are given an undirected graph $G = (V, E)$ and two terminal sets S and T . The task is to find a minimum cut C in G (if there is any) separating S from T under the following “uncut” condition. In the graph $(V, E \setminus C)$, the terminals in each terminal set remain in the same connected component. In spite of the superficial similarity to the classic problem MINIMUM s - t -CUT, TWO-SETS CUT-UNCUT is computationally challenging. In particular, even deciding whether such a cut of *any size* exists, is already NP-complete. We initiate a systematic study of TWO-SETS CUT-UNCUT within the context of parameterized complexity. By leveraging known relations between many well-studied graph parameters, we characterize the structural properties of input graphs that allow for polynomial kernels, fixed-parameter tractability (FPT), and slicewise polynomial algorithms (XP). Our main contribution is the near-complete establishment of the complexity of these algorithmic properties within the described hierarchy of graph parameters.

On a technical level, our main results are fixed-parameter tractability for the (vertex-deletion) distance to cographs and an OR-cross composition excluding polynomial kernels for the vertex cover number of the input graph (under the standard complexity assumption $\text{NP} \not\subseteq \text{coNP/poly}$).

2012 ACM Subject Classification Theory of computation \rightarrow Parameterized complexity and exact algorithms; Theory of computation \rightarrow Problems, reductions and completeness; Mathematics of computing \rightarrow Paths and connectivity problems; Mathematics of computing \rightarrow Graph algorithms; Mathematics of computing \rightarrow Graph coloring

Keywords and phrases Fixed-parameter tractability, Polynomial Kernels, $W[1]$ -hardness, XP, para-NP-Hardness

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.14

Funding *Matthias Bentert*: Supported by the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No. 819416).

Fedor V. Fomin: Supported by the Research Council of Norway via the project BWCA (grant no. 314528).

Fanny Hauser: Supported by the German Academic Exchange Service under the Erasmus+ program (Grant Agreement 2023#009).

Saket Saurabh: Supported by the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No. 819416).



© Matthias Bentert, Fedor V. Fomin, Fanny Hauser, and Saket Saurabh;
licensed under Creative Commons License CC-BY 4.0

19th International Symposium on Parameterized and Exact Computation (IPEC 2024).

Editors: Édouard Bonnet and Paweł Rzażewski; Article No. 14; pp. 14:1–14:23

Leibniz International Proceedings in Informatics



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

1 Introduction

We study TWO-SETS CUT-UNCUT, a natural optimization variant of 2-DISJOINT CONNECTED SUBGRAPHS. In 2-DISJOINT CONNECTED SUBGRAPHS, we are given an undirected graph G and two disjoint sets S, T of vertices. The question is whether there are two disjoint sets R, B of vertices such that $S \subseteq R$, $T \subseteq B$, and both $G[R]$ and $G[B]$ (the graphs induced by R and B , respectively) are connected. 2-DISJOINT CONNECTED SUBGRAPHS is the special case of DISJOINT CONNECTED SUBGRAPHS with two sets of terminals (a problem that played a crucial role in the graph-minors project by Robertson and Seymour [20]). Consequently, 2-DISJOINT CONNECTED SUBGRAPHS has received considerable research attention, particularly from the graph-algorithms and the computational-geometry communities [7, 12, 13, 15, 18, 22]. In TWO-SETS CUT-UNCUT, we not only want to decide whether there are disjoint connected sets containing terminal sets S and T , respectively, but also minimize the size of the corresponding cut (if it exists). Formally, TWO-SETS CUT-UNCUT is defined as follows. Therein, an S - T -cut (R, B) is a partition of the set of vertices into R and B with $S \subseteq R$ and $T \subseteq B$. The set $\text{cut}_G(R)$ contains all edges in G with exactly one endpoint in R (and the other in B).

TWO-SETS CUT-UNCUT

Input: A connected undirected graph $G = (V, E)$, two sets $S, T \subseteq V$, and an integer ℓ .

Question: Is there an S - T -cut (R, B) of G with $|\text{cut}_G(R)| \leq \ell$ such that the vertices of S are in the same connected component of $G[R]$ and the vertices of T are in the same connected component of $G[B]$?

We mention in passing that we assume the input graph to be connected as it becomes trivial if there are at least two connected components containing terminal vertices (vertices in $S \cup T$) and if all terminals belong to one connected component, then we can discard all other connected components. When G is connected, it is also easy to see that any optimal solution for TWO-SETS CUT-UNCUT cuts the graph in exactly two connected components as any connected component not containing any terminal can be merged with any other connected component reducing the size of the cut in the process. Finally, if $S \cap T \neq \emptyset$, then the instance is a trivial no-instance.

Related work. 2-DISJOINT CONNECTED SUBGRAPHS was intensively studied and its complexity is quite well understood. Gray et al. [12] showed that 2-DISJOINT CONNECTED SUBGRAPHS is NP-hard on planar graphs and van 't Hoft et al. [13] showed NP-hardness even if $|S| = 2$ and on P_5 -free split graphs. Note that the problem becomes trivial if $|S| = 1$. Since split graphs are chordal, their results also show that 2-DISJOINT CONNECTED SUBGRAPHS is NP-hard on split graphs. They also complemented the NP-hardness on P_5 -free graphs by providing a polynomial-time algorithm for P_4 -free graphs (also known as cographs). Kern et al. [15] generalized this result by showing that for each graph H , 2-DISJOINT CONNECTED SUBGRAPHS is polynomial-time solvable on H -free graphs if and only if H is a subgraph of a P_4 together with any number of isolated vertices (and otherwise NP-hard). Cygan et al. [7] studied the parameterized complexity with respect to the number $k = n - |S \cup T|$ of non-terminal vertices in the graph. They showed that 2-DISJOINT CONNECTED SUBGRAPHS cannot be solved in $O^*((2 - \varepsilon)^k)$ time for any $\varepsilon > 0$ unless the strong exponential time hypothesis fails. Moreover, they showed that it does not admit a polynomial kernel for this parameter unless $\text{NP} \subseteq \text{coNP/poly}$. As 2-DISJOINT CONNECTED SUBGRAPHS is the special case of TWO-SETS CUT-UNCUT where $\ell = m$, all of the above hardness results directly transfer to TWO-SETS CUT-UNCUT.

The problem TWO-SETS CUT-UNCUT was introduced by Bentert et al. [2] who showed that the problem is $W[1]$ -hard parameterized by $|T|$ even if $|S| = 1$ in general graphs but fixed-parameter tractable when parameterized by $|S \cup T|$ in planar graphs. They also showed fixed-parameter tractability on planar graphs, when parameterized by the minimum size of a set of faces in any planar embedding such that each terminal is incident to one of the faces in the set. Moreover, TWO-SETS CUT-UNCUT is a special case of MIXED MULTIWAY CUT-UNCUT. In this problem, one is not restricted to two sets of terminals and one is given two integers k and ℓ as input. The question is whether one can delete at most k vertices and at most ℓ edges to separate all terminals in different sets while maintaining connectivity within each terminal set. Rai et al. [19] showed that this problem is fixed-parameter tractable when parameterized by $k + \ell$ which immediately implies that TWO-SETS CUT-UNCUT is fixed-parameter tractable when parameterized by ℓ .

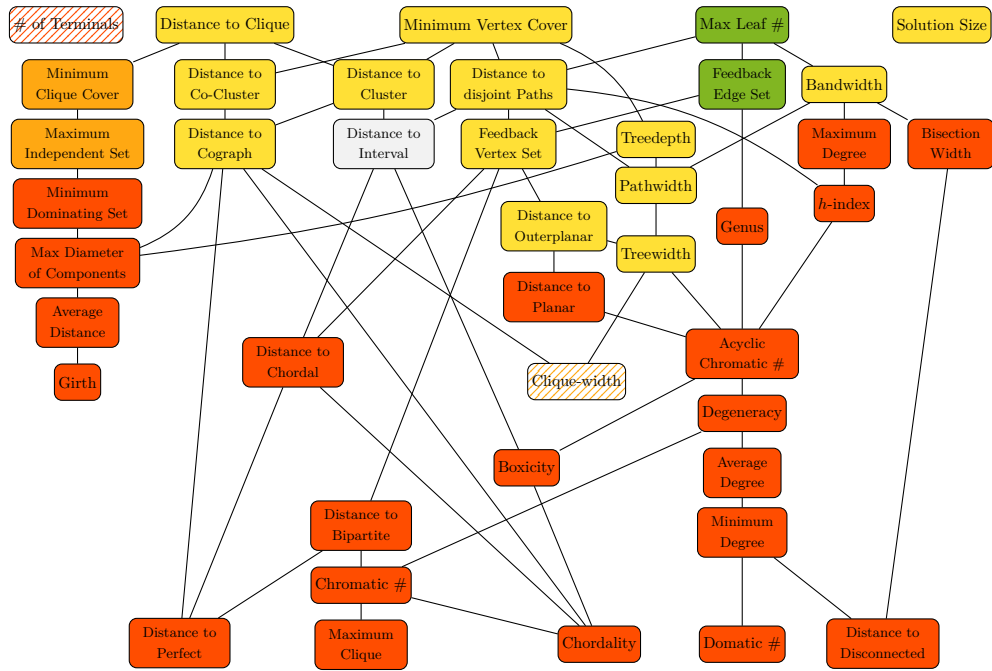
Another related problem is called LARGEST BOND. Here, we are looking for a largest cut that cuts a connected graph into exactly two connected components (and no terminal sets are given). Duarte et al. [9] showed that the problem is NP-hard on bipartite split graphs, fixed-parameter tractable when parameterized by treewidth, does not admit a polynomial kernel when parameterized by the solution size, can be solved in $f(k)n^{O(k)}$ time, where k is the clique-width of the input graph, but not in $f(k)n^{o(k)}$ time unless the exponential time hypothesis fails. In particular, this also excludes fixed-parameter tractability.

Last but not least, TWO-SETS CUT-UNCUT is closely related to NETWORK DIVERSION, which has been studied extensively by the operations-research and networks communities [5, 6, 10, 14, 16]. In this problem, we are given an undirected graph G , two terminal vertices s and t , an edge $b = \{u, v\}$, and an integer ℓ . The question is whether it is possible to delete at most ℓ edges such that the edge b will become a bridge with s on one side and t on the other. Equivalently, is there a minimal s - t -cut of size at most $\ell + 1$ containing b . While this problem seems very similar to the classic MINIMUM s - t -CUT, the complexity status of this problem (polynomial-time solvable or NP-hard) is widely open. This problem is a special case of TWO-SETS CUT-UNCUT where $|S| = |T| = 2$ as there are only two cases. Either s is in the same component as u or s is in the same component as v . These two cases correspond to instances of TWO-SETS CUT-UNCUT with $S = \{s, u\}$ and $T = \{t, v\}$ and $S = \{s, v\}$ and $T = \{t, u\}$, respectively.

Our contribution. We provide an almost complete tetrachotomy for TWO-SETS CUT-UNCUT distinguishing between parameters that allow for polynomial kernels, fixed-parameter tractability, or slice-wise polynomial (XP-time) algorithms. Our results are summarized in Figure 1. The rest of this work is organized as follows. In Section 2, we introduce concepts and notation used throughout the paper. In Section 3, we present fixed-parameter tractable and slice-wise polynomial (XP-time) algorithms. In Section 4, we exclude the possibility for further fixed-parameter tractable or XP-time algorithms by presenting $W[1]$ -hardness and para-NP-hardness results, respectively. Section 5 is devoted to both positive and negative results regarding the existence of polynomial kernels and we conclude with Section 6.

2 Preliminaries

For a positive integer n , let $[n] = \{1, 2, \dots, n\}$. We use standard graph-theoretic terminology and all graphs in this work are undirected. In particular, for an undirected graph $G = (V, E)$ we set $n = |V|$ and $m = |E|$. For a subset $V' \subseteq V$ of the vertices, we use $G[V']$ to denote the subgraph of G induced by V' and denote by $G - V'$ the subgraph $G[V \setminus V']$. Moreover, for



■ **Figure 1** Overview of our results. An edge between two parameters α and β , where α is above β , indicates that in any instance, the value of β is upper-bounded by a function only depending on the value of α . Any hardness result for α immediately implies the same hardness result for β and any positive result for β immediately implies the same positive result for α (where we additionally require that the dependency is polynomial if we show or exclude a polynomial kernel). Green boxes indicate the existence of polynomial kernels, yellow boxes show that the parameter admits fixed-parameter tractability but no polynomial kernel, an orange box indicates polynomial-time algorithms for constant parameter values (XP) but no fixed-parameter tractability, and a red box shows that the parameter is NP-hard for some constant parameter value. We mention that the status of TWO-SETS CUT-UNCUT parameterized by distance to interval graphs, number of terminals (XP/para-NP-hard), and clique-width (fixed-parameter tractable/W[1]-hard) remain open.

an edge set $E' \subseteq E$, we denote by $G - E' = (V, E \setminus E')$ the graph resulting from deleting the edges in E' from G . The *degree* $\deg_G(v)$ of v is the number of vertices adjacent to v in G . A *path* $P = (v_1, v_2, \dots, v_\ell)$ on ℓ vertices is a graph with vertex set $\{v_1, v_2, \dots, v_\ell\}$ and edge set $\{\{v_i, v_{i+1}\} \mid i \in [\ell - 1]\}$. The vertices v_1 and v_ℓ are called *endpoints*. The *length* of a path is its number of edges. A *connected component* in a graph is a maximal set V' of vertices such that for each pair $u, v \in V'$, there is a path in the graph with endpoints u and v . A *cut* in a graph is the set of edges between any partition of the vertices of a graph into two disjoint subsets. A *separation* in a graph $G = (V, E)$ is a pair (X, Y) of sets of vertices with $X \cup Y = V$ and no edges between $X \setminus Y$ and $Y \setminus X$. The size of the separation is $X \cap Y$. The *disjoint union* of two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ results in the graph $(V_1 \cup V_2, E_1 \cup E_2)$. The *join* of G_1 and G_2 results in the graph $(V_1 \cup V_2, E_1 \cup E_2 \cup \{\{u, v\} \mid u \in V_1 \wedge v \in V_2\})$, that is, we first take the disjoint union and then add all possible edges between the two graphs. We refer to the Bachelor’s thesis of Schröder [21] for an overview over how the different parameters are related to one another.

To streamline some of our arguments, we use the following natural reinterpretation of TWO-SETS CUT-UNCUT. The task is to color each vertex in the graph red or blue such that all vertices in S are red, all vertices in T are blue, the graphs induced by the set of all red

vertices (and all blue vertices, respectively) are connected, and there are at most ℓ edges with a red and a blue endpoint. We call such edges multicolored. We often keep sets R and B of red and blue vertices, respectively, and we use the notation $N_X^r(v)$ and $N_X^b(v)$ to denote all red and blue neighbors of v in a set X of vertices, respectively.

Parameterized complexity. A *parameterized problem* is a set of instances (I, k) where $I \in \Sigma^*$ is a problem instance from some finite alphabet Σ and the integer k is the *parameter*. A parameterized problem L is *fixed-parameter tractable* if $(I, k) \in L$ can be decided in $f(k) \cdot |I|^{O(1)}$ time, where f is a computable function only depending on k . We call (I, k) a *yes-instance* (of L) if $(I, k) \in L$. The class XP contains all parameterized problems which can be decided in polynomial time if the parameter k is constant, that is, in $f(k) \cdot |I|^{g(k)}$ time for computable functions f and g . It follows from the definition that each fixed-parameter tractable problem is contained in XP. To show that a problem is not contained in XP, one can show that the problem remains NP-hard for some constant parameter value. To show that a parameterized problem L is presumably not fixed-parameter tractable, one may use a *parameterized reduction* from a W[1]-hard problem to L [8]. A parameterized reduction from a parameterized problem L to another parameterized problem L' is an algorithm that, given an instance (I, k) of L , computes an instance (I', k') of L' in $f(k) \cdot |I|^{O(1)}$ time such that (I, k) is a yes-instance if and only if (I', k') is a yes-instance and $k' \leq g(k)$ for two computable functions f and g . A *kernelization* is an algorithm that, given an instance (I, k) of a parameterized problem L , computes in $|I|^{O(1)}$ time an instance (I', k') of L (the *kernel*) such that (I, k) is a yes-instance if and only if (I', k') is a yes-instance and $|I'| + k' \leq f(k)$ for some computable function f only depending on k . We say that f measures the *size* of the kernel. If f is a polynomial, then we say that P admits a *polynomial kernel*. A problem is fixed-parameter tractable if and only if it admits a kernel of any size. Assuming $\text{NP} \not\subseteq \text{coNP/poly}$, one can show that certain parameterized problems do not admit a polynomial kernel. This can for example be done via OR-cross-compositions. For the definition of OR-cross-compositions, we first need the following. Given an NP-hard problem L , an equivalence relation \mathcal{R} on the instances of L is a *polynomial equivalence relation* if one can decide for any two instances in polynomial time whether they belong to the same equivalence class, and for any finite set S of instances, \mathcal{R} partitions the set into at most $(\max_{I \in S} |I|)^{O(1)}$ equivalence classes.

► **Definition 1** (OR-cross-composition [3]). *Given an NP-hard problem Q , a parameterized problem L , and a polynomial equivalence relation \mathcal{R} on the instances of Q , an OR-cross-composition of Q into L (with respect to \mathcal{R}) is an algorithm that takes t instances I_1, I_2, \dots, I_t of Q belonging to the same equivalence class of \mathcal{R} and constructs in time polynomial in $\sum_{i=1}^t |I_i|$ an instance (I, k) of L such that k is polynomially upper-bounded by $\max_{i \in [t]} |I_i| + \log(t)$ and (I, k) is a yes-instance of L if and only if there exists an $i \in [t]$ such that I_i is a yes-instance of Q .*

If a parameterized problem admits an OR-cross-composition, then it does not admit a polynomial kernel unless $\text{NP} \subseteq \text{coNP/poly}$ [3].

Graph parameters and classes. We give an overview of the different graph parameters and graph classes used throughout the paper. To this end, let $G = (V, E)$ be a graph. The *maximum degree* of G is the largest degree of any vertex in V . The *distance to Π* for some graph class Π is the minimum number of vertices needed to be removed from G such that it becomes a graph in Π . A *cograph* is a graph without induced paths of length three.

Equivalently, a cograph is a graph that can be represented by a cotree. A cotree is a rooted binary tree in which the leaves correspond to the vertices in the cograph and the internal nodes correspond to taking the disjoint union or the join of the cographs corresponding to the two children. A join of two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ results in the graph $(V_1 \cup V_2, E_1 \cup E_2 \cup \{\{u, v\} \mid u \in V_1 \wedge v \in V_2\})$, that is, we first take the disjoint union and then add all possible edges between the two graphs. An *interval graph* is a graph where each vertex can be represented by an interval of real numbers such that two vertices are adjacent if and only if their respective intervals overlap. An *independent set* in a graph is a set of pairwise non-adjacent vertices. The vertex set of a *bipartite graph* can be partitioned into two independent sets. The *vertex cover number* of G is the distance to an independent set. A *clique* in a graph is a set of pairwise adjacent vertices. The *minimum clique cover* of G is the minimum number of cliques needed to partition V . A *dominating set* in a graph is a set of vertices such that each vertex not contained in the set has at least one neighbor in the set. A *tree decomposition* of G is a tree T with nodes X_1, X_2, \dots, X_p , where each X_i is a subset of V such that each vertex in V and both endpoints of each edge in E are contained in some X_i and all X_i that contain some vertex v form one connected component in T . The width of a tree decomposition is the size of its largest set X_i minus one and the *treewidth* of G is the minimum width of any tree decomposition of G . The *clique-width* of G is defined as the minimum number of labels needed to construct G using the following operations in which i and j are some arbitrary labels: Creating a single vertex with label i , the disjoint union of two graphs, adding all possible edges between the vertices of label i and the vertices of label j , and changing the label of all vertices of label i to label j . The *feedback edge number* of G is the minimum size of a set F of edges such that $G - F$ is a forest. Given an injective function f that maps the vertices in V to distinct integers, the bandwidth cost of f for G is defined as $\max_{\{u,v\} \in E} |f(u) - f(v)|$. The *bandwidth* of G is the minimum bandwidth cost for G over all possible injective functions. The *bisection width* of G is the minimum size of a set F of edges such that the vertices of $G - F$ can be partitioned into two parts of equal size (or with a difference of one in case n is odd) with no edges between the two parts.

3 Parameterized Algorithms

In this section, we present some parameterized algorithms for TWO-SETS CUT-UNCUT. We start with the distance to cographs.

► **Theorem 2.** *TWO-SETS CUT-UNCUT parameterized by the distance k to cographs can be solved in $k^{O(k)}n^3$ time.*

Proof. Let $(G = (V, E), S, T, \ell)$ be an instance of TWO-SETS CUT-UNCUT. We transform G into a weighted graph by assigning a weight of one to each edge in G . We denote the weight of an edge e by $w(e)$. We first compute in $O(3.303^k(m+n))$ time a set $X \subseteq V$ of size at most k such that $G' = (V', E') = G - X$ is a cograph [17]. A cotree \mathcal{T} of G' is a rooted binary tree where each vertex of G' corresponds to a leaf node of \mathcal{T} and an inner node t of \mathcal{T} either represents taking the disjoint union or the join of the cographs corresponding to the two children of t . Each cograph has a cotree which can be computed in linear time. For each node t of \mathcal{T} , let \mathcal{T}_t be the subtree of \mathcal{T} rooted in t , let $G_t = (V_t, E_t)$ be the graph represented by \mathcal{T}_t , and let $n_t = |V_t|$. For technical reasons, we want to assume that X contains at least one red and one blue vertex in an optimal solution. Hence, if X does not contain a terminal from S already, then we add an arbitrary vertex from S to X . We do the same for T .

We begin by guessing¹ the coloring of X . Then, we remove all edges between a red and a blue vertex in X . Let ℓ' be the number of edges that we removed. We then contract each component of X into a single vertex. If the resulting (multi)graph has j edges with the same endpoints (with one endpoint in X and one in V'), then we remove all but one of the edges and set its weight to j .

We compute a coloring for the remaining vertices of G by computing optimal solutions for partial instances for each node of \mathcal{T} via dynamic programming. A partial instance is a tuple (t, r, C_r, C_b) where t is a node of \mathcal{T} , r is an integer at most n_t and C_r, C_b are partitions of subsets of X including \emptyset . For each partial instance, we want to store in a table D the minimum number of edges between blue and red vertices in $G[V_t \cup X]$ after all vertices in V_t are colored and exactly r vertices are colored red. Additionally, we require that for each set $C \in C_r$, there exists a connected component with vertex set Z in $G[(V_t \cup X) \cap R]$ such that $Z \cap X \cap R = C$. Analogously for each set $C \in C_b$, there has to exist a connected component with vertex set Z in $G[(V_t \cup X) \cap B]$ such that $Z \cap X \cap B = C$. We will use these sets to store whether the red vertices are connected in $G[V_t \cup X]$, while also storing which vertices of $X \cap R$ are connected to the same connected component in $G_t[R]$. We use this information to ensure that in a coloring for the entire graph G , the graph $G[R]$ is connected. We do the same for the blue vertices with the set C_b .

In the end, the optimal solution (minus the ℓ' edges we already removed between vertices in X) will be stored in $D[w, r, C_r, C_b]$ for some value of r , where w is the root of \mathcal{T} and $C_r = \emptyset$ if $r = 0$ and $C_r = \{X \cap R\}$, otherwise. Similarly, $C_b = \emptyset$ if $r = n_w$ and $C_b = \{X \cap B\}$, otherwise. Note that this corresponds to a solution where all vertices of either color form a single connected component as all vertices are connected to all vertices of the same color in X (which is at least one vertex as constructed above).

Before we present the algorithm, we first define two operations on sets of sets. For two sets A, B of sets of vertices, if there exists $a \in A$ and $b \in B$ with $a \cap b \neq \emptyset$, then we recursively define $A \uplus B$ as $((A \setminus a) \uplus (B \setminus b)) \uplus \{a \cup b\}$ and as $A \cup B$ otherwise. This operation can be seen as taking the union of the connected components of two subgraphs. If there are two connected components (one in A and one in B) which share a vertex, then both components are merged into one. The components that do not share a vertex with any other component remain as they are. We also define $A \uplus B$ as A if $B = \emptyset$, as B if $A = \emptyset$ and as $\{\bigcup_{X \in A \cup B} X\}$, otherwise. With these definitions at hand, we compute the entries of D based on the type of node t as follows.

Leaves. Let t be a leaf node and let a be the vertex of G corresponding to t . We set

$$D[t, r, C_r, C_b] = \begin{cases} \sum_{v \in N_X^b(a)} w(\{a, v\}), & \text{if } r = 1, C_r = \{N_X^r(a)\}, C_b = \emptyset \text{ and } a \notin T \\ \sum_{v \in N_X^r(a)} w(\{a, v\}), & \text{if } r = 0, C_r = \emptyset, C_b = \{N_X^b(a)\} \text{ and } a \notin S \\ \infty, & \text{else.} \end{cases}$$

We show that the solutions for all partial instances are computed correctly. It is easy to verify that whenever a table entry of D is set, then this corresponds to a valid solution for the partial instance. So it remains to show that each optimal solution for any partial instance is considered. Let (t, r, C_r, C_b) be a partial instance and consider an optimal solution for

¹ Whenever we pretend to guess something, we actually iterate over all possibilities and consider for the presentation/proof an iteration leading to an optimal solution.

it. Note that the partial instance only has a valid solution if $r \in \{0, 1\}$. If $r = 1$, then the vertex a has to be colored red. This is only possible if $a \notin T$ since all vertices in T have to be colored blue. The red vertices of X which are adjacent to vertices of $G_t[R]$ are the vertices in $N_X^r(a)$. If a is not adjacent to any red vertices of X , then $C_r = \{\emptyset\}$ or there is no solution to the partial instance. Since G_t cannot have any blue vertices, C_b has to be the empty set (and not the set containing the empty set). Note that all edges between a and blue neighbors of a in X are multicolored and this is precisely what we computed above. The argument for $r = 0$ (coloring a blue) is symmetric.

Disjoint union. Let t be a disjoint-union node with children t_1 and t_2 . We set

$$D[t, r, C_r, C_b] = \min_{\substack{r_1 \\ C_r = C_{r_1} \uplus C_{r_2} \\ C_b = C_{b_1} \uplus C_{b_2}}} (D[t_1, r_1, C_{r_1}, C_{b_1}] + D[t_2, r - r_1, C_{r_2}, C_{b_2}]).$$

We again show that each optimal solution for any partial instance is considered. To this end, we assume that the table entries for the children t_1 and t_2 are computed correctly. Let (t, r, C_r, C_b) be a partial instance and consider an optimal solution for it. Since the disjoint union of two graphs does not create any edges, it holds that all multicolored edges in the solution are contained in $G[V_{t_1} \cup X]$ and $G[V_{t_2} \cup X]$. This gives a partitioning of the multicolored edges in the solution as there are no edges between vertices in X . Let r_1 be the number of red vertices in V_{t_1} in the solution and consider the connected components in $G[(V_{t_1} \cup X) \cap R]$. Denote the respective partitioning by C_{r_1} and do the same for C_{b_1}, C_{r_2} , and C_{b_2} . Note that C_r corresponds to the union of the connected components corresponding to C_{r_1} and C_{r_2} (and the same for C_b). Hence, it holds that $C_r = C_{r_1} \uplus C_{r_2}$ and $C_b = C_{b_1} \uplus C_{b_2}$. It now also holds that the size of the considered solution is $D[t_1, r_1, C_{r_1}, C_{b_1}] + D[t_2, r - r_1, C_{r_2}, C_{b_2}]$, which is precisely what we computed.

Join. Let t be a join node with children t_1, t_2 . We set (with $r_2 = r - r_1$)

$$D[t, r, C_r, C_b] = \min_{\substack{r_1 \\ C_r = C_{r_1} \uplus C_{r_2} \\ C_b = C_{b_1} \uplus C_{b_2}}} \{D[t_1, r_1, C_{r_1}, C_{b_1}] + D[t_2, r_2, C_{r_2}, C_{b_2}] + r_1(n_2 - r_2) + r_2(n_1 - r_1)\}.$$

We show a final time that each optimal solution for any partial instance is considered when the table entries for the children t_1 and t_2 are computed correctly. Let (t, r, C_r, C_b) be a partial instance and consider an optimal solution for it. Similar to the disjoint union, we can partition all multicolored edges of the solution. In this case, we partition the multicolored edges into four parts, the edges in $G[V_{t_1} \cup X]$, $G[V_{t_2} \cup X]$, the newly created edges between red vertices in V_{t_1} and blue vertices in V_{t_2} and similarly between blue vertices in V_{t_1} and red vertices in V_{t_2} . Let r_1 be the number of red vertices in V_{t_1} in the solution and again consider the connected components in $G[(V_{t_1} \cup X) \cap R]$. Denote the respective partitioning by C_{r_1} and do the same for C_{b_1}, C_{r_2} , and C_{b_2} . If there is at least one red vertex in each of V_{t_1} and V_{t_2} , then all red vertices of G_t will be connected in $G_t[R]$. Thus the red vertices of X which were connected to any vertex of $G_t[R]$ will be in the same set in C_r . If either V_{t_1} or V_{t_2} do not contain any red vertices in the considered solution, then no edges between two red vertices are added and the connected components of $G_t[R]$ are the same as of $G_{t_1}[R] \cup G_{t_2}[R]$ (as at least one of the two sets is the empty set). This is precisely what the \uplus operator computes and the argument for the blue vertices is analogous. Hence, the optimal solution is $D[t_1, r_1, C_{r_1}, C_{b_1}] + D[t_2, r - r_1, C_{r_2}, C_{b_2}] + r_1(n_2 - (r - r_1)) + (r - r_1)(n_1 - r_1)$, which is what we compute in the dynamic program.

It remains to analyze the running time. Note that both \uplus and \upcup can be computed in $O(k^2)$ time. The number of possible guesses for the coloring of X is at most 2^k . The number of partial instances is at most $O(n^2(k+2)^{k+2})$ as there are at most n possibilities for t and r each and both C_r and C_b are partitions of subsets of X which have size at most $k+2$ (as we added a terminal of S and of T to X). Computing each entry takes $O((k+2)^{4(k+2)+2} \cdot n)$ time. Thus, the overall runtime is in $O(2^k \cdot (k+2)^{6k+14} \cdot n^3) \subseteq k^{O(k)} \cdot n^3$. ◀

We next show fixed-parameter tractability for the parameter treewidth. We mention that the algorithm is a simple adaptation of a dynamic program for LARGEST BOND [9]. In essence, each entry in the dynamic program over the tree decomposition (except for the leaves) are computed by combining the solutions for the children using a maximum over all combinations of solutions that color the vertices in the given bag in a certain way and ensure certain connectivity conditions. By replacing the maximum with a minimum, we can solve a version of TWO-SETS CUT-UNCUT without terminals (which is not a hard problem to solve). However, we can also incorporate terminals by setting the value of all leaf nodes corresponding to a vertex v to infinity whenever $v \in S$ and the given coloring of the bag colors v blue (or analogously $v \in T$ and the coloring for v is red). This yields the following.

► **Observation 3.** *TWO-SETS CUT-UNCUT is solvable in $nk^{O(k)}$ time when parameterized by treewidth k .*

We now turn towards XP-time algorithms, that is, polynomial-time algorithms for constant parameter values. We show that TWO-SETS CUT-UNCUT parameterized by the maximum size of an independent set is in XP. The main idea is to first observe that the graphs induced by the vertices in S (and in T , respectively) cannot have too many connected components as this would imply a large independent set in the input graph. Next, these connected components cannot be too far apart from one another as any long induced path contains a large independent set. Based on these two observations, it is enough to guess a small number of vertices to ensure connectivity between all vertices in S and in T , respectively. For each guess, we can then compute a minimum cut between the vertices we already colored red and blue to find an optimal solution.

► **Proposition 4.** *TWO-SETS CUT-UNCUT parameterized by the size k of a maximum independent set in the input graph can be solved in $O(n^{4k^2})$ time.*

Proof. Let $(G = (V, E), S, T, \ell)$ be an instance of TWO-SETS CUT-UNCUT where G has a maximum independent set of size k . Let S_1, S_2, \dots, S_p be the connected components of $G[S]$ and let T_1, T_2, \dots, T_q be the connected components of $G[T]$. Note that $p \leq k$ and $q \leq k$ as we can otherwise choose one vertex from each component to get an independent set of size $k+1$, a contradiction. In the beginning only the vertices of S are colored red and the vertices of T are colored blue. We claim that in any solution, at most $(k-1)(2k-2)$ additional red vertices are needed to connect all vertices of S and at most $(k-1)(2k-2)$ additional blue vertices are needed to connect all vertices of T . To show this, consider any solution. To connect two connected components S_i and S_j of $G[S]$, there has to exist a path between S_i and S_j such that all vertices of the path are colored red. Note that this also implies that there is an induced path between them where all vertices are colored red. It is now possible to bound the length of this path by the size of the maximum independent set. Any induced path between S_i and S_j contains at most $2k$ vertices as any induced path of length $2k+1$ contains an independent set of size $k+1$. Hence, to connect S_i and S_j in any solution at

14:10 The Parameterized Complexity Landscape of Two-Sets Cut-Uncut

most $2k - 2$ additional red vertices are needed. Since there are p connected components and $p \leq k$, at most $(k - 1)(2k - 2)$ vertices have to be colored red to make $G[R]$ connected. The argument for the vertices of T is analogous.

Our algorithm now guesses two sets $R', B' \subseteq V$, each of size at most $(k - 1)(2k - 2)$. We discard any guess where $T \cap R' \neq \emptyset$ or $S \cap B' \neq \emptyset$. Additionally, we discard all guesses where $G[S \cup R']$ or $G[T \cup B']$ are not connected. Finally, we color all remaining vertices of V by computing a minimum $S \cup R' - T \cup B'$ -cut. If there are at most ℓ multicolored edges in G , then (G, ℓ) is a yes-instance.

We next show that the running time is in $O(n^{4^k})$. Note that for $k = 1$, we do not need to guess any vertices and therefore the running time is $O(m^{1+o(1)}) \subseteq O(n^4)$ as we only need to find a minimum cut [4]. For $k \geq 2$, we need to guess $2(k - 1)(2k - 2) = 4k^2 - 8k + 4 \leq 4k^2 - 4$ vertices. Hence, there are at most $n^{4k^2 - 4}$ possible guesses and for each guess finding a minimum cut can be done in $m^{1+o(1)} \in O(n^4)$ time. Thus, the overall runtime is in $O(n^{4k^2})$. ◀

Finally, we show that TWO-SETS CUT-UNCUT parameterized by clique-width is in XP. We note that it remains open whether this parameter allows for fixed-parameter tractability. Our algorithm is an adaptation of an algorithm for LARGEST BOND due to Duarte et al. [9]. They use dynamic programming over a k -expression of the input graph where they store the size of a largest cut with exactly s_i red vertices of each label i under certain connectivity conditions. As in the case of treewidth, we can replace a maximum in their calculation by a minimum to solve a version of TWO-SETS CUT-UNCUT without any terminals. We can then incorporate terminals by first modifying the k -expression into a $3k$ -expression where all vertices in S of label i get label i_S instead and all vertices in T of label i get label i_T instead. We then simply discard any table entry in which the number of red vertices of label i_S does not equal the number of vertices of label i_S or where there are any red vertices of label i_T . Since their algorithm runs in $O(n^{2k+4}3^{6 \cdot 2^k})$ time, this yields the following.

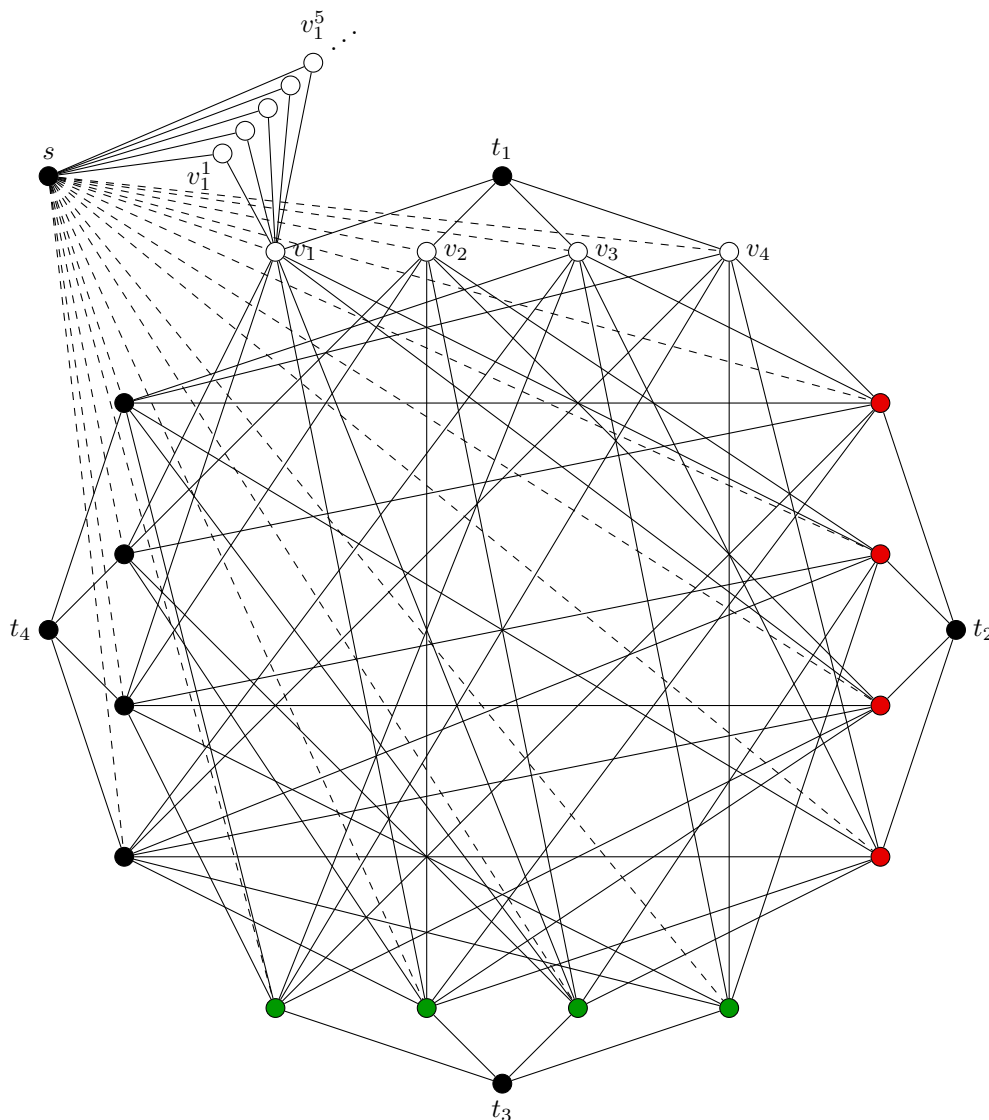
► **Observation 5.** *TWO-SETS CUT-UNCUT can be solved in $O(n^{6k+4}3^{6 \cdot 8^k})$ time when parameterized by the clique-width k .*

4 Parameterized Hardness

In this section, we show a number of hardness (both para-NP-hardness and W[1]-hardness) results for TWO-SETS CUT-UNCUT and 2-DISJOINT CONNECTED SUBGRAPHS. As 2-DISJOINT CONNECTED SUBGRAPHS is a special case of TWO-SETS CUT-UNCUT, all hardness results for the former directly translate to the latter. First, 2-DISJOINT CONNECTED SUBGRAPHS remains NP-hard even in bipartite graphs of bisection width one. The idea for the reduction is to first add a copy of the graph (without any terminals) and connect one vertex of the graph to a vertex in S . Note that all vertices in the copy can be colored red and hence the size of an optimal solution remains the same. Next, we can make the graph bipartite by subdividing each edge once. This results in an equivalent bipartite instance of bisection width one and shows the following.

► **Observation 6.** *2-DISJOINT CONNECTED SUBGRAPHS is NP-hard in bipartite graphs with bisection width one.*

Next, we show that TWO-SETS CUT-UNCUT is W[1]-hard when parameterized by the clique cover number even if the size of a smallest dominating set is one. Our reduction is based on a reduction by Bentert et al. [2] and we prove a couple of additional properties of the reduction that will be useful when excluding polynomial kernels for the solution size.



■ **Figure 2** Illustration of the reduction by Bentert et al.. The dashed edges represent $n + 2m$ parallel P_3 's as indicated between s and v_1 .

► **Proposition 7.** *TWO-SETS CUT-UNCUT is $W[1]$ -hard when parameterized by the clique cover number of the input graph even if the input graph contains a dominating set of size one, when $|S| = 1$, and when there exist constants $c_1 \geq 0, c_2 > |T|$ such that (i) $\ell = c_1 + |T|(c_2 - |T|)$ and (ii) any cut that keeps any set of j terminals in T connected to at least one other terminal in T while separating the terminal in S from all terminals in T has size at least $c_1 + j(c_2 - j)$.*

Proof. Our proof is based on a reduction by Bentert et al. [2]. We first summarize their reduction, which shows that TWO-SETS CUT-UNCUT is $W[1]$ -hard when parameterized by the number of terminals even if $|S| = 1$. Starting from an instance (G, k) of REGULAR MULTICOLORED CLIQUE, a version of MULTICOLORED CLIQUE where each vertex in the input graph has degree exactly d for some d , they construct an equivalent instance (H, S, T, ℓ) of TWO-SETS CUT-UNCUT as shown next and depicted in Figure 2. They start with an

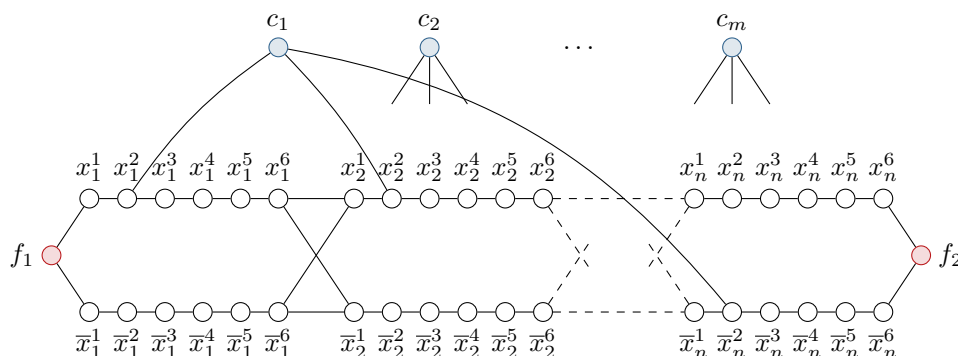
14:12 The Parameterized Complexity Landscape of Two-Sets Cut-Uncut

induced copy of G , set $S = \{s\}$ and $T = \{t_1, t_2, \dots, t_k\}$, and make t_i adjacent to all vertices of color i in G . Moreover, they add vertices v_i^j for each vertex v_i in G and each $j \in [n + 2m]$, where n and m are the number of vertices and edges in G , respectively. Each vertex v_i^j is made adjacent to v_i and s . Finally, they set $\ell = n - k + k(n + 2m) + k(d - k + 1)$.

We next modify the construction to make s adjacent to all vertices in the induced copy of G in H and adjacent to all vertices in T . Next, we assume that there are the same number n/k of vertices of each color in G as this version of the problem is also known to be NP-hard. We further modify the reduction to make each vertex v_i in H adjacent to all other vertices that have the same color (in G) and all vertices of the form v_i^j pairwise adjacent. We adjust the value of $\ell = 2n + k(n + 2m + d - k + 1)$. Note that the new graph H has a dominating set of size one (the vertex s is adjacent to everything) and has a clique cover number of at most $k + 1$. The latter holds as all vertices of one color in G form a clique together with one vertex from T and all other vertices (s and all vertices v_i^j) form one clique C_s . Moreover, observe that $\ell = c_1 + |T|(c_2 - |T|)$ for $c_1 = 2n$ and $c_2 = (n + 2m + d + 1)$.

It remains to prove that the adjusted instance is equivalent to the input instance of REGULAR MULTICOLORED CLIQUE and that any cut that keeps a set of j terminals in T connected to at least one other terminal in T while separating all terminals in T from s has size at least $c_1 + j(c_2 - j)$. We start with the latter. First note that any cut of size at most ℓ cannot cut through the clique C_s . Moreover, in order for any set of j vertices $T' \subseteq T$ to be connected to at least one other vertex in T , we need to add at least one neighbor of t_i to the connected component of t_i for each $t_i \in T'$. Since the neighborhoods of two vertices t_i and t_j are by construction disjoint, this also implies that we need to separate at least j of the original vertices in G from C_s . Moreover, any such cut is minimized when we pick exactly one neighbor of each t_i as each neighbor is adjacent to $n + 2m$ vertices in C_s and only $n/k + d < n + 2m$ vertices outside of C_s . So consider any cut that separates all of T and j additional vertices from C_s . Observe that the size of the cut is at least $n + k - j + j(n + 2m + d - j + 1 + n/k)$ as the vertices in T are incident to $n + k$ edges and all but j of them are in the cut. Moreover, each of the j additional vertices are adjacent to $n/k - 1$ vertices of the same color (which are not separated from C_s), at least $d - j + 1$ vertices of different colors that are not separated from C_s , and $n + 2m + 1$ vertices in C_s . Since we may assume that $n/k > k$ (otherwise there is a simple k^k time algorithm), it holds that $n + k - j + j(n + 2m + d - j + 1 + n/k) \leq 2n + j(n + 2m + d + 1 - j) = c_1 + j(c_2 - j)$.

To show that the constructed instance is equivalent to the original instance of REGULAR MULTICOLORED CLIQUE, we closely follow the proof by Bentert et al. [2]. If the constructed instance is a yes-instance, then we separate exactly one neighbor of each vertex in T from C_s as shown above. Let C be the set of these vertices. In order for a cut between $T \cup C$ and the rest of the graph to be of size at most ℓ , we show that C needs to induce a clique in G . First, each vertex in T is incident to exactly n/k edges in the cut, that is, the vertices in T are incident to exactly n edges in the cut. This leaves a remaining budget of $n + k(n + 2m + d - k + 1)$ for edges incident to the vertices in C . Hence, each of these vertices can be adjacent to $(n + 2m + d - k + 1 + n/k)$ edges in the cut on average. Note that every vertex is adjacent to exactly $n + 2m + 1$ vertices in C_s and to exactly $n/k - 1$ vertices of the same color (which are all not separated from C_s). Hence, on average each vertex in C can be incident to at most $d - k + 1$ edges to vertices of other colors in G that are not separated from C_s . As all of these edges are exactly the edges in G and since G is d -regular, each vertex needs to be incident to $k - 1$ edges to other vertices in C , that is, C needs to be a clique of size k .



■ **Figure 3** An illustration of the reduction behind Proposition 8 with $C_1 = (x_1 \vee x_2 \vee \bar{x}_n)$.

If there is a (multicolored) clique C of size k in G , then consider the cut in H between $T \cup C$ and the rest of the graph. Each vertex in T is incident to $n/k + 1$ edges and adjacent to one other vertex in $T \cup C$. Hence, the cut contains $k(n/k)$ edges incident to vertices in T . Moreover, it contains $k(n/k - 1 + d - (k - 1))$ edges between the original vertices in G in H and $k(n + 2m + 1)$ edges between vertices in C and vertices in C_s . In total, the cut has size $k(2n/k + d - k + n + 2m + 1) = 2n + k(n + 2m + d - k + 1) = \ell$. This concludes the proof. ◀

Finally, we show that 2-DISJOINT CONNECTED SUBGRAPHS remains NP-hard on graphs of maximum degree three based on a reduction due to van 't Hof et al. [13]. Note that both 2-DISJOINT CONNECTED SUBGRAPHS and TWO-SETS CUT-UNCUT become trivial on graphs of maximum degree at most 2 as the input graph is then restricted to a path or a cycle.

► **Proposition 8.** *2-DISJOINT CONNECTED SUBGRAPHS is NP-hard even if the input graph has maximum degree three.*

Proof. We give a reduction from 3,4-SAT, which is known to be NP-hard [23]. Therein, one is given a Boolean formula ϕ in which each clause contains at most three literals and each variable appears in at most four clauses. Let $X = \{x_1, x_2, \dots, x_n\}$ and $\bar{X} = \{\bar{x} \mid x \in X\}$ be a set of all positive and negative literals in ϕ and let $C = \{C_1, C_2, \dots, C_m\}$ be the set of clauses in ϕ . We construct an instance (G, S, T) of 2-DISJOINT CONNECTED SUBGRAPHS as follows. For each clause C_i , we add a vertex c_i to G and for each variable x_i , we add paths $P_i = (x_i^1, x_i^2, \dots, x_i^6)$ and $\bar{P}_i = (\bar{x}_i^1, \bar{x}_i^2, \dots, \bar{x}_i^6)$ to G . If a literal $x_i \in X \cup \bar{X}$ appears in the clause C_j , then we add an edge between c_j and one vertex of $\{x_i^2, x_i^3, x_i^4, x_i^5\}$ which is not adjacent to any other vertex $c_{j'}$. Such a vertex always exists as each variable appears in at most four clauses. Finally, we add edges $\{x_i^6, \bar{x}_{i+1}^1\}$, $\{\bar{x}_i^6, x_{i+1}^1\}$, $\{x_i^6, x_{i+1}^1\}$ and $\{\bar{x}_i^6, \bar{x}_{i+1}^1\}$ for all $i \in [n - 1]$ and vertices f_1 and f_2 with edges $\{f_1, x_1^1\}$, $\{f_1, \bar{x}_1^1\}$, $\{f_2, x_n^6\}$ and $\{f_2, \bar{x}_n^6\}$. We conclude the construction by setting $S = \{f_1, f_2\}$ and $T = C$. See Figure 3 for an illustration.

We next show that the maximum degree in G is three. Note that each vertex c_i has degree at most three as each clause in ϕ contains at most three variables. For each path P_i , the vertices x_i^j with $2 \leq j \leq 4$ have degree at most 3 because each x_i^j is only adjacent to x_i^{j-1} and x_i^{j+1} and at most one vertex of C . The vertices x_i^1 and x_i^6 have degree 3 since they are not adjacent to any vertices in C and only have edges to $x_{i-1}^6, \bar{x}_{i-1}^6, x_i^2$ and $x_{i+1}^1, \bar{x}_{i+1}^1, x_i^5$, respectively (or to f_1 or f_2 in the case of $x_1^1, \bar{x}_1^1, x_n^6$, and \bar{x}_n^6). The vertices f_1 and f_2 each only have degree two and the entire graph has therefore maximum degree three.

Since the reduction can clearly be computed in polynomial time, it only remains to show that the constructed instance is a yes-instance of 2-DISJOINT CONNECTED SUBGRAPHS if and only if ϕ is satisfiable. To this end, first assume that the constructed instance is a yes-instance and let (R, B) be a coloring of G such that $G[R]$ and $G[B]$ are connected and $S \subseteq R$ and $T \subseteq B$. We set each variable x_i to true if and only if any vertex of P_i is colored blue. We next prove that this is a satisfying assignment. The vertices f_1 and f_2 are both colored red. Any path that connects f_1 to f_2 in $G[R]$ cannot contain any vertices of C as $C = T$. Hence, any path between f_1 and f_2 has to contain all vertices of P_i or all vertices of \bar{P}_i for each $i \in [n]$. As a result, all vertices of P_i or all vertices of \bar{P}_i are colored red. Since the vertices of $C = T$ are independent, one neighbor x of each vertex c_j has to be colored blue by the solution. If $x = x_i^p$ for some $i \in [n]$ and $2 \leq p \leq 5$, then we set x_i by construction to true and c_j is satisfied. If $x = \bar{x}_i^p$ for some $i \in [n]$ and $2 \leq p \leq 5$, then coloring one vertex of \bar{P}_j blue forces all vertices of P_j to be colored red and thus x_i was by construction set to false. The vertex c_i is only adjacent to a vertex of \bar{P}_j if \bar{x}_i appears in C_i and therefore setting x_j to false satisfies C_i . This proves that ϕ is satisfiable.

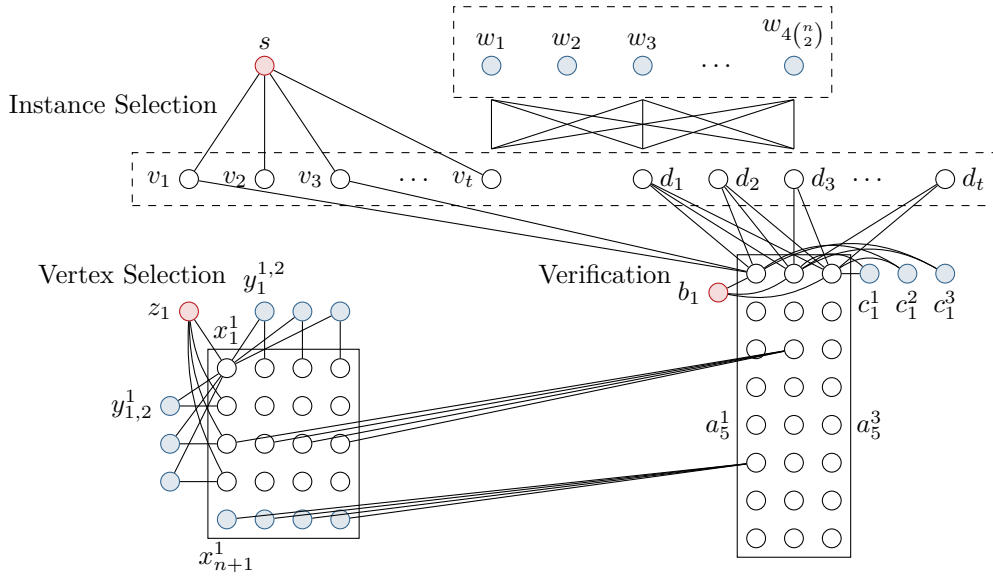
Now assume that there is a truth assignment β that satisfies ϕ . We color all vertices of P_i blue and all vertices of \bar{P}_i red if x_i is set to true by β and we color all vertices of P_i red and all vertices of \bar{P}_i blue, otherwise. Next, we color the vertices of C blue and f_1 and f_2 red. In this coloring exactly one path of P_i and \bar{P}_i is colored red and the other is colored blue. All paths P_i and \bar{P}_i are connected to $P_{i-1}, \bar{P}_{i-1}, P_{i-1}$, and \bar{P}_{i+1} (which the exception of P_1, \bar{P}_1 and P_n, \bar{P}_n which are connected to f_1 and f_2 instead). Hence, all red vertices in the paths are connected and all blue vertices in the paths are connected as well. The vertices f_1 and f_2 are by construction adjacent to one red vertex in one of the paths. Finally for each $c_i \in C$, since β satisfies C_i , there has to be a blue neighbor of c_i in one of the paths. This shows that both $G[R]$ and $G[B]$ are connected in the constructed coloring and this concludes the proof. ◀

5 Polynomial Kernels

In this section, we analyze which parameters allow for polynomial kernels. Note that we can restrict our attention to parameters that allow for fixed-parameter tractability as this is equivalent to having a kernel of any size [8]. We first show that TWO-SETS CUT-UNCUT does not admit a polynomial kernel when parameterized by the vertex cover number of the input graph plus the number of terminals. Note that for this parameter, 2-DISJOINT CONNECTED SUBGRAPHS has a simple kernel of size $O(k^3)$. We can 2-approximate a vertex cover in polynomial time by repeatedly taking both endpoints of any uncovered edge into the solution. We can then, for each pair of vertices in this vertex cover, mark k non-terminal vertices in the common neighborhood (or all such vertices if there are less than k). Removing all unmarked non-terminal vertices that do not belong to the approximated vertex cover results in a cubic kernel as we keep at most $2k$ vertices in the approximate vertex cover and at most $(2k)^2 \cdot k = 4k^3$ marked vertices.

► **Theorem 9.** *TWO-SETS CUT-UNCUT parameterized by vertex cover number plus number of terminals does not admit a polynomial kernel unless $\text{NP} \subseteq \text{coNP/poly}$.*

Proof. We present an OR-cross composition from VERTEX COVER in 3-regular graphs, which is known to be NP-hard [11]. We start with t instances, all with the same number n of vertices and the same solution size k . Note that since all graphs have the same number of vertices and are 3-regular, they also have the same number m of edges.



■ **Figure 4** An illustration of the construction in the proof of Theorem 9. The solid boxes indicate cliques and the dashed boxes indicate independent sets.

Our OR-cross composition consists of three different gadgets. The first gadget is called the instance-selection gadget and it consists of one vertex v_j for each input instance and t additional dummy vertices D . These $2t$ vertices all form an independent set. Next, we add a vertex s and make it adjacent to all vertices v_j and a set W of $4\binom{n}{2}$ vertices and make them adjacent to each vertex in the instance-selection gadget. We add s to S and W to T .

The second gadget is called a vertex-selection gadget. We start with a clique of $(n+1) \cdot k$ vertices x_i^j for each $i \in [n+1]$ and each $j \in [k]$. All vertices x_{n+1}^j are contained in T . Next, for each pair x_i^p and x_i^q with $p \neq q \in [k]$ and each $i \in [n]$, we add a vertex $y_i^{p,q}$, add it to T and make it adjacent to x_i^p and x_i^q . We do the same for each pair x_p^j and x_q^j for each $p \neq q \in [n]$ and each $j \in [k]$ and call the constructed terminal $y_{p,q}^j$. Finally, for each $j \in [k]$, we create a vertex z_j , add it to S , and make it adjacent to each vertex x_i^j for $i \in [n]$.

The third and final gadget is called the verification gadget and it consists of a clique of $3\binom{n}{2}$ vertices a_i^j for $i \in [\binom{n}{2}]$ and $j \in [3]$. For each $i \in [\binom{n}{2}]$, we add four additional vertices b_i, c_i^1, c_i^2, c_i^3 . The vertex b_i is added to S and the other three are added to T . We make b_i adjacent to all vertices a_i^j for $j \in [3]$. Moreover, c_i^1 is adjacent to a_i^1 and a_i^2 , vertex c_i^2 is adjacent to a_i^1 and a_i^3 , and c_i^3 is adjacent to a_i^2 and a_i^3 .

It remains to connect the different gadgets. We arbitrarily order the vertices in each input instance and assign them numbers in $[n]$. Next, we pick an arbitrary bijection f between numbers in $[\binom{n}{2}]$ and pairs $\{p, q\}$ with $p \neq q \in [n]$. For each $i \in [\binom{n}{2}]$, the vertex a_i^1 is adjacent to v_j if and only if $f(i)$ is *not* an edge in the instance corresponding to v_j . Moreover, a_i^1 is adjacent to some arbitrary vertices in D to ensure that it has exactly t neighbors in the instance-selection gadget. The vertices a_i^2 and a_i^3 are adjacent to the t vertices in D .

Next, for each $i \in [\binom{n}{2}]$, we make a_i^1 adjacent to x_{n+1}^j in the vertex-selection gadget for all $j \in [k]$. Let $f(i) = \{p, q\}$ with $p < q$. Then, we make a_i^2 adjacent to x_p^j and a_i^3 adjacent to x_q^j for all $j \in [k]$. Finally, we set $\ell = 2\binom{n}{2}^2 + \binom{n}{2}(t+k+7) + k^2n + k(3n+2k-4) + t - m - 1$.

Note that the instance-selection gadget contains an independent set of size $2t$ which does not contain any terminals. There are only $1 + 4\binom{n}{2} + k(n+2) + k\binom{n}{2} + n\binom{k}{2} + 7\binom{n}{2} < 14n^3$ other vertices. Thus, the vertex cover number and the number of terminals of the resulting

14:16 The Parameterized Complexity Landscape of Two-Sets Cut-Uncut

graph is in $O(n^3)$. Since the instance can be computed in polynomial time (in $n + t$), it only remains to show that the constructed instance is a yes-instance if and only if at least one of the input instances of VERTEX COVER is a yes-instance.

To this end, first assume that one of the t instances of VERTEX COVER is a yes-instance, that is, the respective graph G_j contains a vertex cover $K = \{p_1, p_2, \dots, p_k\}$ of size k . Let v_j be the vertex in the instance-selection gadget corresponding to that instance. We show that the constructed instance is also a yes-instance by describing the connected component C_S containing S . This component contains all the vertices of S as well as v_j , $x_{p_i}^i$ for each $i \in [k]$, and the following vertices of the verification gadget. For each $i \in \binom{[n]}{2}$, if $f(i) = \{p, q\}$ (with $p < q$) is not an edge in G_j , then we add a_i^1 to C_S . Otherwise, at least one of the endpoints p or q is contained in K and we add a_i^2 if $p \in K$ and a_i^3 if $p \notin K$. Note that we added exactly one vertex of each column in the vertex-selection gadget and exactly one vertex in each row of the verification gadget. It remains to show that the cut between C_S and the rest of the graph is of size at most ℓ .

We first analyze the sizes of cuts within each of the gadgets. Note s is incident to exactly t edges and exactly one of the neighbors (v_j) is contained in C_S . Hence, s is incident to $t - 1$ edges in the cut between C_S and the rest of the graph. Next, observe that v_j is incident to $4\binom{[n]}{2}$ vertices in W (which are not contained in C_S). Each vertex z_j in the vertex-selection gadget is incident to n vertices exactly one of which is in C_S . Each of the other k vertices in C_S in the vertex-selection gadget are adjacent to $((n+1)k - 1) + (n-1) + (k-1) + 1$ other vertices in the vertex-selection gadget, exactly k of which belong to C_S . Next, each vertex b_i is adjacent to exactly two vertices not contained in C_S and each vertex a_i^j in C_S is adjacent to exactly $2\binom{[n]}{2} + 2$ vertices in the verification gadget that are not contained in C_S .

We next analyze the number of edges in the cut between C_S and the rest of the graph that go between different gadgets. Note that there are no edges between the instance-selection gadget and the vertex-selection gadget. Hence, we only need to consider edges leaving the verification gadget. We start with the size of such a cut assuming that no vertex in the verification gadget belongs to C_S . Note that v_j has $\binom{[n]}{2} - m$ edges to the verification gadget and each of the k vertices x_i^j in the vertex-selection gadget that belong to C_S have $n - 1$ edges to the verification gadget. This leads to a baseline cut of size $\binom{[n]}{2} - m + k(n - 1)$. Now notice that adding any vertex in the verification gadget to C_S increases the described cut by $(t + k)$ if none of the neighbors of the vertex in other gadgets are contained in C_S and by $t + k - 2$ if one neighbor belongs to C_S . By construction, it never happens for a vertex in the verification gadget that two neighbors outside the verification gadget are contained in C_S . Moreover, we constructed the solution such that one neighbor is always contained in C_S . Hence, the overall size of the described cut is $\binom{[n]}{2} - m + k(n - 1) + \binom{[n]}{2}(t + k - 2)$. Combined with the size of the cuts within each gadget, the total cut size is

$$\begin{aligned} & t - 1 - m + \binom{[n]}{2}(t - k + 5) + k(nk + 3n + 2k - 4) + \binom{[n]}{2}(2\binom{[n]}{2} + 2) \\ &= 2\binom{[n]}{2}^2 + \binom{[n]}{2}(t + k + 7) + k^2n + k(3n + 2k - 4) + t - m - 1 = \ell. \end{aligned}$$

Note that both C_S and the rest of the graph induce a single connected component each.

For the reverse direction, suppose that the constructed instance of TWO-SETS CUT-UNCUT is a yes-instance. Let $C_S \supseteq S$ be the set of vertices in the connected component containing S after removing the edges of a solution (a cut of size at most ℓ). First, we will argue that we can assume without loss of generality that $C_S \setminus S$ contains exactly one vertex v_j , exactly one vertex from the set $\{a_i^1, a_i^2, a_i^3\}$ for each $i \in \binom{[n]}{2}$, exactly one vertex from the

set $\{x_1^j, x_2^j, \dots, x_n^j\}$ for each $j \in [k]$, and at most one vertex from each set $\{x_i^1, x_i^2, \dots, x_i^k\}$ for each $i \in [n]$. Note that this also implies that $C_S \setminus S$ contains exactly $\binom{n}{2} + k + 1$ vertices as all other vertices belong to S or T . Assume that $C_S \setminus S$ contains at least two vertices from the instance-selection gadget. If connectivity within C_S is not of concern, then removing one of the two vertices from C_S will always decrease the size of the cut as each vertex in the instance-selection gadget is adjacent to at most $3\binom{n}{2}$ vertices in C_S but also to $4\binom{n}{2}$ vertices in W (which are contained in T and therefore not in C_S). Moreover, since all vertices in the verification gadget that have neighbors in the instance-selection gadget (all a -vertices) form a clique and vertices in the instance-selection gadget are only incident to such vertices and s , two vertices are never required to ensure connectivity within C_S . On the other hand, at least one vertex from the vertex-selection gadget needs to be contained in C_S in order to connect $s \in S$ with the rest of S .

Next, assume that two vertices from a set $\{a_i^1, a_i^2, a_i^3\}$ are contained in C_S . Then, these two vertices have a common neighbor c_i^j for some $j \in [3]$ which is contained in T but only has these two neighbors. This contradicts the fact that we started with some solution to TWO-SETS CUT-UNCUT. Again, at least one such vertex needs to be included in C_S in order to connect $b_i \in S$ with the rest of S . The same arguments apply to the sets $\{x_1^j, x_2^j, \dots, x_n^j\}$ and $\{x_i^1, x_i^2, \dots, x_i^k\}$ with the exception that there is no vertex in S enforcing that at least one vertex of $\{x_i^1, x_i^2, \dots, x_i^k\}$ belongs to C_S .

We next show that the vertices K encoded by the set of vertices in $C_S \setminus S$ in the vertex-selection gadget form a vertex cover in the instance corresponding to the vertex $v_j \in C_S \setminus S$ in the instance-selection gadget. As in the forward direction, the size of the cut between C_S and the rest of the graph has size at least $\ell = 2\binom{n}{2}^2 + \binom{n}{2}(t+k+7) + k^2n + k(3n+2k-4) + t - m - 1$ and this bound is only achieved if each vertex in $C_S \setminus S$ in the verification gadget (each a vertex in C_S) has exactly one neighbor in $C_S \setminus S$ outside the verification gadget. We call such a neighbor the *buddy* of the vertex. For each $i \in [\binom{n}{2}]$, if a_i^1 is in C_S and has a buddy, then this buddy must be v_j indicating that $f(i)$ is not an edge in v_j . If a_i^2 or a_i^3 is contained in C_S and has a buddy, then at least one of the endpoints of $f(i)$ is contained in K . This implies that for each pair $\{p, q\}$ of vertices it holds that $\{p, q\}$ is not an edge in the instance corresponding to v_j or p or q is contained in K , that is, K is a vertex cover in this instance. Note that the set K has size k as $C_S \setminus S$ does not contain two vertices from the set $\{x_1^j, x_2^j, \dots, x_n^j\}$ for any $j \in [k]$. This concludes the proof. \blacktriangleleft

We can make the instance-selection gadget into a clique in the above proof without changing any of the proof details except for the fact that the size of an optimal solution increases by exactly $2t - 1$ (as we still need to include exactly one vertex of the vertex-selection gadget in the connected component containing S). This gives the following.

► **Corollary 10.** *TWO-SETS CUT-UNCUT parameterized by distance to clique plus the number of terminals does not admit a polynomial kernel unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.*

We next show that TWO-SETS CUT-UNCUT admits a linear kernel in the feedback edge number of the graph by simple data reduction rules for vertices of degree one and two.

► **Proposition 11.** *TWO-SETS CUT-UNCUT parameterized by feedback edge number k admits a kernel with at most $5k$ vertices and $6k$ edges.*

Proof. We present data-reduction rules that eliminate all degree-1 vertices and bound the length of maximal induced paths, that is, paths whose internal vertices have degree two in the input graph. Using standard arguments, this will result in a kernel with $O(k)$ vertices and edges [1].

14:18 The Parameterized Complexity Landscape of Two-Sets Cut-Uncut

First, note that since the input graph is connected, we can assume without loss of generality that there are no isolated vertices (vertices without any neighbors). Next, consider a vertex u with exactly one neighbor v . If $u \notin S \cup T$, then we can simply remove u as in any optimal solution, we will color u with the same color as v and therefore u will not be incident to any solution edges. If $u \in S \cup T$ (we assume without loss of generality in S), then there are two cases. If $S = \{u\}$, then the instance is trivial as if $T = \emptyset$, then it is a yes-instance and if $T \neq \emptyset$, then the instance is a yes-instance if and only if $\ell \geq 1$ as deleting the edge $\{u, v\}$ is an optimal solution (as the input graph is connected). If $|S| \geq 2$, then u needs to be in the same connected component as v in any optimal solution. Hence, we can remove u and add v to S instead. If v was already contained in T , then we return a trivial no-instance.

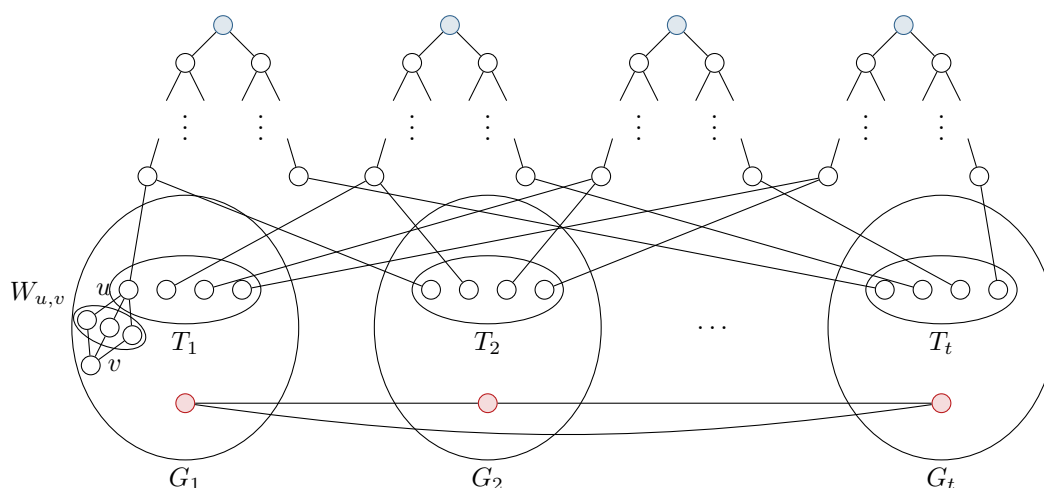
After applying the above procedure exhaustively, we are left with a graph without any vertices of degree at most one. Consider any maximal induced path, that is, a path $P = (v_0, v_1, \dots, v_p)$ for some p such that $\deg_G(v_0), \deg_G(v_p) > 2$ and $\deg_G(v_i) = 2$ for all $i \in [p-1]$. Let $I = \{v_i \mid i \in [p-1]\}$ be the set of internal vertices of P . We consider the following cases based on whether $I \cap S = \emptyset$ and $I \cap T = \emptyset$. If $I \cap S = \emptyset$ and $I \cap T = \emptyset$, then we remove all vertices in I except for v_1 and add the edge $\{v_1, v_p\}$. Let $P' = (v_0, v_1, v_p)$ be the resulting maximal induced path. Note that if v_0 and v_p are colored with the same color, then the internal vertices of P can be colored with the same color and therefore do not increase the solution size. If v_0 and v_p are colored differently, then we need to remove exactly one of the edges in P and this also holds true for P' . If $I \cap S \neq \emptyset$ and $I \cap T = \emptyset$, then we again replace P by P' but this time we add v_1 to S . In this case, we can assume without loss of generality that all vertices in I are colored red as we need to delete at least one edge for each of v_0 and v_p that are colored blue and we can assume without loss of generality that these edges are $\{v_0, v_1\}$ and/or $\{v_{p-1}, v_p\}$. The case where $I \cap S = \emptyset$ and $I \cap T \neq \emptyset$ is analogous. Finally, assume that $I \cap S \neq \emptyset$ and $I \cap T \neq \emptyset$. Then let $v_i \in S$ and $v_j \in T$ such that no vertex between v_i and v_j is contained in $S \cup T$. Note that such a pair of vertices always exists. We assume without loss of generality that $i < j$. Since some edge between v_i and v_j has to belong to any solution, we can remove the edge $\{v_i, v_{i+1}\}$ without loss of generality and reduce ℓ by one. Note that both v_i and v_{i+1} now have degree one and applying the above procedure for degree-1 vertices exhaustively removes all vertices in I (or produces a trivial kernel).

Hence, we can now assume that the graph does not contain any degree-1 vertices and all maximal induced paths have at most one internal vertex. Using standard arguments, the graph contains now at most $5k$ vertices and at most $6k$ edges, where k is the feedback edge number of the constructed graph [1, Lemma 2]. Note that the feedback edge number of the graph never increases in the above procedures. Hence, the produced instance has at most $5k$ vertices and $6k$ edges, where k is the feedback edge number of the input graph. This concludes the proof. \blacktriangleleft

We next show that the solution size does not allow for a polynomial kernel. Therein, we use Proposition 7 to construct an OR-cross composition in which a solution to the entire instance can only consist of a solution to one of the input instances.

► **Proposition 12.** *TWO-SETS CUT-UNCUT parameterized by solution size ℓ does not admit a polynomial kernel unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.*

Proof. We present an OR-cross composition from TWO-SETS CUT-UNCUT, where $|S| = 1$ and there exist constants $c_1 \geq 0, c_2 > |T|$ such that $\ell = c_1 + |T|(c_2 - |T|)$ and any cut that keeps any set of j terminals in T connected to at least one other terminal in T while

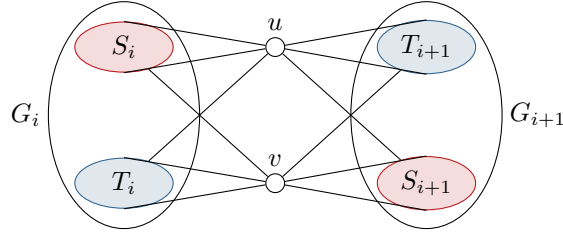


■ **Figure 5** An illustration of the reduction behind Proposition 12.

separating the terminal in S from all terminals in T has size at least $c_1 + j(c_2 - j)$. This variant is NP-hard by Proposition 7. We consider the polynomial equivalence relation where all instances in the same equivalence class have the same solution size ℓ' , the same constants c_1, c_2 , and the same number $k = |T|$ of terminals in T . We assume that the number t of input instances is 2^q for some integer q . Note that this can be achieved by duplicating one of the input instances at most t times.

Given t instances $I_1 = (G_1, S_1, T_1, \ell')$, $I_2 = (G_2, S_2, T_2, \ell')$, \dots , $I_t = (G_t, S_t, T_t, \ell')$ of the same equivalence class, we construct an instance (H, S, T, ℓ) of TWO-SETS CUT-UNCUT as follows. We start with H being the disjoint union of all graphs G_i and S being the union of all S_i . Let $w = k \log(t) + 1$. We replace each edge $\{u, v\}$ in the current graph by w paths of length two, that is, we add a set $W_{u,v}$ of w new vertices and for each $p \in W_{u,v}$, we add the edges $\{u, p\}$ and $\{p, v\}$. Next, we add k binary trees of depth $\log(t)$, add the roots of these trees to T , and identify the i^{th} leaf of each of these trees with one vertex in T_i in such a way that each vertex in T_i is identified with the leaf of exactly one tree. See Figure 5 for an illustration. Finally, we add edges between each pair of vertices in S and set $\ell = w\ell' + k \log(t)$. This concludes the construction.

Since the solution size ℓ is by construction polynomially upper-bounded by $n + \log(t)$ and the reduction can be computed in polynomial time (in $n + t$), it only remains to show that the constructed instance is a yes-instance if and only if at least one of the input instances is a yes-instance. To this end, first assume that one input instance I_i is a yes-instance. We construct a solution of size at most ℓ in the constructed instance as follows. For each edge $\{u, v\}$ in the solution, we add all edges between vertices in $W_{u,v}$ and u to our solution cut. Note that this cut has size at most $w\ell'$. Next, for each of the k binary trees, we separate the path from the root to the vertex in G_i from the rest of the tree. Note that this cuts exactly one edge in each layer and hence exactly $\log(t)$ edges per tree. Thus, the constructed cut has size at most $w\ell' + k \log(t) = \ell$. Note that all vertices in T (the roots of the binary trees) are connected to one another through the graph G_i and are separated from all other graphs G_j with $j \neq i$ as they are separated from all other leaves in the binary tree and within G_i they are separated from S_i . Since all vertices in S are connected by construction, this shows that the constructed instance is a yes-instance.



■ **Figure 6** A connection gadget in the proof of Proposition 13.

For the reverse direction, assume that the constructed instance is a yes-instance and let F be a solution cut of size at most ℓ in H . First note that if for some pair u, v of vertices, an edge incident to a vertex in $W_{u,v}$ is contained in F , then we can assume without loss of generality that for each vertex in $W_{u,v}$ exactly one incident edge is cut. Since $w > k \log(t)$, this means that we can assume that F contains edges incident to at most ℓ' sets $W_{u,v}$. Let F' be the set of edges in the t original graphs corresponding to these ℓ' sets. Note that if some set of j terminals in T are connected through some graph G_i in $H - F$, then they are connected through a path in their respective binary trees and the j corresponding terminals in G_i have to be separated from S_i . By assumption, such a cut (in the original graph G_i) has size at least $c_1 + j(c_2 - j)$ and in H , this corresponds to a cut of size $w(c_1 + j(c_2 - j))$. In order to connect all terminal pairs, we have to connect all k through one graph G_i as if we use at least two graphs to connect sets of size j_1, j_2, \dots, j_p with $\sum_{i=1}^p j_i \geq k$, then this cut has size at least

$$\sum_{i=1}^p w(c_1 + j_i(c_2 - j_i)) \geq \sum_{i=1}^p w(c_1 + j_i(c_2 - k)) \geq w(pc_1 + k(c_2 - k)) \geq w(c_1 + 1 + k(c_2 - k)) > \ell,$$

a contradiction. If all terminals are connected through a single graph G_i , then the vertices in T_i remain connected to one another in $H - F$ and are separated from S_i . That is, there is a set of ℓ' edges in G_i to separate all vertices in T_i from S_i while keeping all vertices in T_i connected. Since $|S_i| = 1$, this shows that instance I_i is a yes-instance of TWO-SETS CUT-UNCUT and this concludes the proof. ◀

Finally, we show that 2-DISJOINT CONNECTED SUBGRAPHS does not admit a polynomial kernel parameterized by bandwidth.

► **Proposition 13.** *2-DISJOINT CONNECTED SUBGRAPHS parameterized by bandwidth does not admit a polynomial kernel unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.*

Proof. We present an OR-cross composition from 2-DISJOINT CONNECTED SUBGRAPHS. Given t instances $I_1 = (G_1, S_1, T_1), I_2 = (G_2, S_2, T_2), \dots, I_t = (G_t, S_t, T_t)$, we construct a new instance (H, S, T) as follows. Initially H is the disjoint union of all G_i and S and T are the unions of all S_i and T_i respectively, where we put graph G_{i+1} to the right of graph G_i . We then use the gadget depicted in Figure 6 to connect G_i and G_{i+1} for all $i \in [t - 1]$. We call this gadget a *connection gadget* and it simply consists of two vertices u and v . We make u adjacent to all vertices in S_i and T_{i+1} as well as one arbitrary vertex in T_i and one arbitrary vertex in S_{i+1} . The vertex v is adjacent to all vertices in T_i and S_{i+1} as well as one arbitrary vertex in each of S_i and T_{i+1} . This concludes the construction.

Since the instance can be computed in polynomial time, it only remains to show that the bandwidth of the constructed instance is polynomial in the maximum number n of vertices in any of the instances and that the constructed instance is a yes-instance if and only if at least

one of the input instances is a yes-instance. For the former, note that placing all vertices of G_1 (in any ordering) first, then the two vertices of the connection gadget between G_1 and G_2 , then all vertices of G_2 and so on results in an ordering where each edge within one of the graphs has length at most $n - 1$ and each edge incident to a vertex in the connection gadget has length at most $n + 1$. Since this covers all edges in the constructed instance, this shows that the bandwidth is upper-bounded by $n + 1$.

We next show that the constructed instance is a yes-instance if and only if one of the input instances is a yes-instance. To this end, first assume that one instance I_i is a yes-instance. We construct a cut in the constructed instance as follows. We start with the solution cut in G_i that leaves both S_i and T_i connected and separates the two within G_i . Next, for the connection gadget between G_j and G_{j+1} for all $j \geq i$, we color u blue and v red. For all other connection gadgets, we color u red and v blue. Finally, in each graph G_j with $i \neq j$, we compute a minimum S_j - T_j -cut (ignoring connectivity) and add it to the solution. Note that all terminals in S_j and T_j for all $j \neq i$ are connected to one another by a vertex in a connection gadget and also connected to one vertex of the same color in the next graph G_{j+1} . Hence, the constructed instance is a yes-instance.

For the reverse direction, assume that the constructed instance is a yes-instance. Note that in each connection gadget, any optimal solution colors both vertices of the gadget differently as otherwise at least one set of terminals is separated. Consider the set of connection gadgets in which u is colored red. If this set is empty, then we consider I_1 and otherwise we consider the instance directly to the right of the last connection gadget in the set. We claim that the considered instance I_i is a yes-instance. As we assume that the u vertex in the connection gadget between G_{i-1} and G_i is colored red (if it exists) and the u vertex in the connection gadget between G_i and G_{i+1} is colored blue (again assuming it exists), each vertex in a connection gadget is adjacent to one terminal of the same color in G_i , that is, they do not provide any additional connectivity between vertices in S_i and T_i . Hence, the solution for the constructed instance contains a cut in G_i such that all vertices of S_i remain connected, are separated from all vertices in T_i , which in turn remain connected to one another. That is, the instance I_i is a yes-instance. This concludes the proof. \blacktriangleleft

6 Conclusion



In this work, we studied the parameterized complexity of TWO-SETS CUT-UNCUT, a natural optimization variant of 2-DISJOINT CONNECTED SUBGRAPHS. We gave an almost complete tetrachotomy in terms of the existence of polynomial kernels, fixed-parameter tractability, and XP-time algorithms. We conclude with a couple of open questions. First, the complexity with respect to the distance to interval graphs remains unclear, with everything between fixed-parameter tractability and para-NP-hardness still being possibilities. In particular, the complexity of TWO-SETS CUT-UNCUT on interval graphs (polynomial-time solvable or NP-hard) is unknown. Second, we showed that there is an XP-time algorithm for the parameter clique-width. Moreover, it is known that LARGEST BOND is W[1]-hard when parameterized by the clique-width. We conjecture that the same holds true for TWO-SETS CUT-UNCUT. Finally, it is known that TWO-SETS CUT-UNCUT is W[1]-hard parameterized by the number of terminals (vertices in $S \cup T$) [2]. However, it is not known whether there is an XP-time algorithm and in particular, even whether NETWORK DIVERSION, a special case of TWO-SETS CUT-UNCUT with four terminals, is polynomial-time solvable or not has been a long-standing open question, which we repeat here.

References

- 1 Matthias Bentert, Alexander Dittmann, Leon Kellerhals, André Nichterlein, and Rolf Niedermeier. An adaptive version of Brandes' algorithm for betweenness centrality. *Journal of Graph Algorithms and Applications*, 24(3):483–522, 2020. doi:10.7155/JGAA.00543.
- 2 Matthias Bentert, Pål Grønås Drange, Fedor V. Fomin, Petr A. Golovach, and Tuukka Korhonen. Two-sets cut-uncut in planar graphs. In *Proceedings of the 51st International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 22:1–22:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ICALP.2024.22.
- 3 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM Journal on Discrete Mathematics*, 28(1):277–305, 2014. doi:10.1137/120880240.
- 4 Jan van den Brand, Li Chen, Richard Peng, Rasmus Kyng, Yang P. Liu, Maximilian Probst Gutenberg, Sushant Sachdeva, and Aaron Sidford. A deterministic almost-linear time algorithm for minimum-cost flow. In *Proceedings of the 64th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 503–514. IEEE, 2023. doi:10.1109/FOCS57990.2023.00037.
- 5 Christopher Cullenbine, R. Kevin Wood, and Alexandra M. Newman. Theoretical and computational advances for network diversion. *Networks*, 62(3):225–242, 2013. doi:10.1002/NET.21514.
- 6 Norman D. Curet. The network diversion problem. *Military Operations Research*, 6(2):35–44, 2001.
- 7 Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. Solving the 2-disjoint connected subgraphs problem faster than 2^n . *Algorithmica*, 70(2):195–207, 2014. doi:10.1007/S00453-013-9796-X.
- 8 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013. doi:10.1007/978-1-4471-5559-1.
- 9 Gabriel L. Duarte, Hiroshi Eto, Tesshu Hanaka, Yasuaki Kobayashi, Yusuke Kobayashi, Daniel Lokshantov, Lehilton L. C. Pedrosa, Rafael C. S. Schouery, and Uéverton S. Souza. Computing the largest bond and the maximum connected cut of a graph. *Algorithmica*, 83(5):1421–1458, 2021. doi:10.1007/S00453-020-00789-1.
- 10 Ozgur Erken. *A branch-and-bound algorithm for the network diversion problem*. PhD thesis, Naval Postgraduate School, 2002.
- 11 Herbert Fleischner, Gert Sabidussi, and Vladimir I. Sarvanov. Maximum independent sets in 3- and 4-regular Hamiltonian graphs. *Discrete Mathematics*, 310(20):2742–2749, 2010. doi:10.1016/J.DISC.2010.05.028.
- 12 Chris Gray, Frank Kammer, Maarten Löffler, and Rodrigo I. Silveira. Removing local extrema from imprecise terrains. *Computational Geometry: Theory and Applications*, 45(7):334–349, 2012. doi:10.1016/J.COMGEO.2012.02.002.
- 13 Pim van 't Hof, Daniël Paulusma, and Gerhard J. Woeginger. Partitioning graphs into connected parts. *Theoretical Computer Science*, 410(47-49):4834–4843, 2009. doi:10.1016/J.TCS.2009.06.028.
- 14 Benjamin S. Kallemyn. *Modeling Network Interdiction Tasks*. PhD thesis, Air Force Institute of Technology, 2015.
- 15 Walter Kern, Barnaby Martin, Daniël Paulusma, Siani Smith, and Erik Jan van Leeuwen. Disjoint paths and connected subgraphs for H -free graphs. *Theoretical Computer Science*, 898:59–68, 2022. doi:10.1016/J.TCS.2021.10.019.
- 16 Chungmok Lee, Donghyun Cho, and Sungsoo Park. A combinatorial Benders decomposition algorithm for the directed multiflow network diversion problem. *Military Operations Research*, 24(1):23–40, 2019.
- 17 James Nastos and Yong Gao. Bounded search tree algorithms for parametrized cograph deletion: Efficient branching rules by exploiting structures of special graph classes. *Discrete Mathematics, Algorithms and Applications*, 4(1), 2012. doi:10.1142/S1793830912500085.

- 18 Daniël Paulusma and Johan M. M. van Rooij. On partitioning a graph into two connected subgraphs. *Theoretical Computer Science*, 412(48):6761–6769, 2011. doi:10.1016/J.TCS.2011.09.001.
- 19 Ashutosh Rai, M. S. Ramanujan, and Saket Saurabh. A parameterized algorithm for mixed-cut. In *Proceedings of the 12th Latin American Symposium on Theoretical Informatics LATIN*, pages 672–685. Springer, 2016. doi:10.1007/978-3-662-49529-2_50.
- 20 Neil Robertson and Paul D. Seymour. Graph minors XIII: The disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63(1):65–110, 1995. doi:10.1006/JCTB.1995.1006.
- 21 Johannes C. B. Schröder. Comparing graph parameters. Bachelor’s thesis, Technische Universität Berlin, 2019.
- 22 Jan Arne Telle and Yngve Villanger. Connecting terminals and 2-disjoint connected subgraphs. In *Proceedings of the 39th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 418–428. Springer, 2013. doi:10.1007/978-3-642-45043-3_36.
- 23 Craig A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85–89, 1984. doi:10.1016/0166-218X(84)90081-7.

Preprocessing to Reduce the Search Space for Odd Cycle Transversal

Bart M. P. Jansen  

Eindhoven University of Technology, The Netherlands

Yosuke Mizutani  

School of Computing, University of Utah, Salt Lake City, UT, USA

Blair D. Sullivan  

School of Computing, University of Utah, Salt Lake City, UT, USA

Ruben F. A. Verhaegh  

Eindhoven University of Technology, The Netherlands

Abstract

The NP-hard ODD CYCLE TRANSVERSAL problem asks for a minimum vertex set whose removal from an undirected input graph G breaks all odd cycles, and thereby yields a bipartite graph. The problem is well-known to be fixed-parameter tractable when parameterized by the size k of the desired solution. It also admits a randomized kernelization of polynomial size, using the celebrated matroid toolkit by Kratsch and Wahlström. The kernelization guarantees a reduction in the total *size* of an input graph, but does not guarantee any decrease in the size of the solution to be sought; the latter governs the size of the search space for FPT algorithms parameterized by k . We investigate under which conditions an efficient algorithm can detect one or more vertices that belong to an optimal solution to ODD CYCLE TRANSVERSAL. By drawing inspiration from the popular *crown reduction* rule for VERTEX COVER, and the notion of *antler decompositions* that was recently proposed for FEEDBACK VERTEX SET, we introduce a graph decomposition called *tight odd cycle cut* that can be used to certify that a vertex set is part of an optimal odd cycle transversal. While it is NP-hard to compute such a graph decomposition, we develop parameterized algorithms to find a set of at least k vertices that belong to an optimal odd cycle transversal when the input contains a tight odd cycle cut certifying the membership of k vertices in an optimal solution. The resulting algorithm formalizes when the search space for the solution-size parameterization of ODD CYCLE TRANSVERSAL can be reduced by preprocessing. To obtain our results, we develop a graph reduction step that can be used to simplify the graph to the point that the odd cycle cut can be detected via color coding.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Fixed parameter tractability

Keywords and phrases odd cycle transversal, parameterized complexity, graph decomposition, search-space reduction, witness of optimality

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.15

Related Version *Full Version:* <https://arxiv.org/abs/2409.00245> [12]

Funding *Blair D. Sullivan:* Gordon & Betty Moore Foundation under grant GBMF4560.

1 Introduction

The NP-hard ODD CYCLE TRANSVERSAL problem asks for a minimum vertex set whose removal from an undirected input graph G breaks all odd cycles, and thereby yields a bipartite graph. Finding odd cycle transversals has important applications, for example in computational biology [8, 21] and adiabatic quantum computing [6, 7]. ODD CYCLE TRANSVERSAL parameterized by the desired solution size k has been studied intensively,



© Bart M. P. Jansen, Yosuke Mizutani, Blair D. Sullivan, and Ruben F. A. Verhaegh; licensed under Creative Commons License CC-BY 4.0

19th International Symposium on Parameterized and Exact Computation (IPEC 2024).

Editors: Édouard Bonnet and Paweł Rzażewski; Article No. 15; pp. 15:1–15:18

Leibniz International Proceedings in Informatics



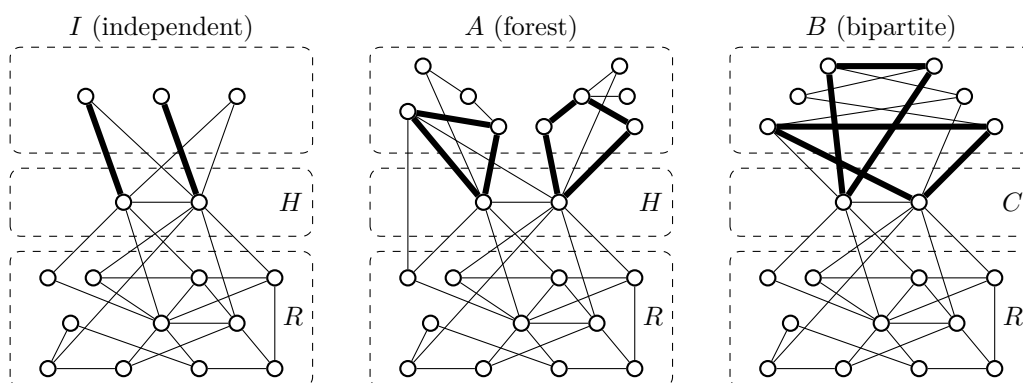
LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

leading to important advances such as *iterative compression* [19] and *matroid-based kernelization* [14, 15]. The randomized kernel due to Kratsch and Wahlström [15, Lemma 7.11] is a polynomial-time algorithm that reduces an n -vertex instance (G, k) of ODD CYCLE TRANSVERSAL to an instance (G', k') on $\mathcal{O}((k \log k \log \log k)^3)$ vertices, that is equivalent to the input instance with probability at least 2^{-n} . Experiments with this matroid-based kernelization, however, show disappointing preprocessing results in practice [18]. This formed one of the motivations for a recent line of research aimed at preprocessing that reduces the *search space* explored by algorithms solving the reduced instance, rather than preprocessing aimed at reducing the *encoding size* of the instance (which is captured by kernelization). To motivate our work, we present some background on this topic.

A *kernelization* of size $f: \mathbb{N} \rightarrow \mathbb{N}$ for a parameterized problem \mathcal{P} is a polynomial-time algorithm that reduces any parameterized instance (x, k) to an instance (x', k') with the same YES/NO answer, such that $|x'|, k' \leq f(k)$. It therefore guarantees that the size of the instance is reduced in terms of the complexity parameter k . It does not directly ensure a reduction in the *search space* of the follow-up algorithm that is employed to solve the reduced instance. Since the running times of FPT algorithms for the natural parameterization of ODD CYCLE TRANSVERSAL [8, 19, 16] depend exponentially on the size of the sought solution, the size of the search space considered by such algorithms can be reduced significantly by a preprocessing step that finds some vertices S that belong to an optimal solution for the input graph G : the search for a solution of size k on G then reduces to the search for a solution of size $k - |S|$ on $G - S$. Researchers therefore started to investigate in which situations an efficient preprocessing phase can guarantee finding part of an optimal solution.

One line of inquiry in this direction aims at finding vertices that not only belong to an optimal solution, but are even required for building a c -approximate solution [2, 13]; such vertices are called *c-essential*. This has resulted in refined running time guarantees, showing that an optimal odd cycle transversal of size k can be found in time $2.3146^{k-\ell} \cdot n^{\mathcal{O}(1)}$, where ℓ is the number of vertices in the instance that are essential for making a 3-approximate solution [2]. Another line of research, more relevant to the subject of this paper, aims at finding vertices that belong to an optimal solution when there is a simple, locally verifiable certificate of the existence of an optimal solution containing them. So far, the latter direction has been explored for VERTEX COVER (where a *crown decomposition* [1, 5] forms such a certificate), and for the (undirected) FEEDBACK VERTEX SET problem (where an *antler decomposition* [4]) forms such a certificate.

A *crown decomposition* (see Figure 1) of a graph G consists of a partition of its vertex set into three parts: the *crown* I (which is required to be a non-empty independent set), the *head* H (which is required to contain all neighbors of I), and the *remainder* $R = V(G) \setminus (I \cup H)$, such that the graph $G[I \cup H]$ contains a matching M of size $|H|$. Since I is an independent set, this matching partners each vertex of H with a private neighbor in I . The existence of a crown decomposition shows that there is an optimal vertex cover (a minimum-size vertex set intersecting all edges) that contains all vertices of H and none of I : any vertex cover contains at least $|M| = |H|$ vertices from $I \cup H$ to cover the matching M , while H covers all the edges of G that can be covered by selecting vertices from $I \cup H$. Hence a crown decomposition forms a polynomial-time verifiable certificate that there is an optimal vertex cover containing all vertices of H . It facilitates a reduction in search space for VERTEX COVER: graph G has a vertex cover of size k if and only if $G - (I \cup H)$ has one of size $k - |H|$. A crown decomposition can be found in polynomial time if it exists, which yields a powerful reduction rule for VERTEX COVER [1].



■ **Figure 1** Examples of crown decomposition (left), antler decomposition for FEEDBACK VERTEX SET (middle) and a tight OCC for ODD CYCLE TRANSVERSAL (right). Packings of forbidden subgraphs are highlighted in bold.

Inspired by this decomposition for VERTEX COVER, Donkers and Jansen [4] introduced the notion of an *antler decomposition* of a graph G . It is a partition of the vertex set into three parts: the *antler* A (which is required to induce a non-empty acyclic graph), the *head* H (which is required to contain *almost* all neighbors of A : for each tree T in the forest $G[A]$, there is at most one edge that connects T to a vertex outside H), and the *remainder* $R = V(G) \setminus (A \cup H)$, while satisfying an additional condition in terms of an integer z that represents the *order* of the antler decomposition. In its simplest form for $z = 1$ (we discuss $z > 1$ later), the additional condition says that the graph $G[A \cup H]$ should contain $|H|$ vertex-disjoint cycles. Since $G[A]$ is acyclic, each of these cycles contains exactly one vertex of H . They certify that any feedback vertex set of G contains at least $|H|$ vertices from $A \cup H$. Since A induces an acyclic graph, and all cycles in G that enter a tree T of $G[A]$ from R must leave A from H , the set H intersects all cycles of G that contain a vertex of $A \cup H$. Hence there is an optimal feedback vertex set containing H . By finding an antler decomposition we can therefore reduce the problem of finding a size- k solution in G to finding a size- $(k - |H|)$ solution in $G - (A \cup H)$, and therefore reduce the search space for algorithms parameterized by solution size.

Donkers and Jansen proved that, assuming $P \neq NP$, there unfortunately is no polynomial-time algorithm to find an antler decomposition if one exists [4, Theorem 3.4]. However, they gave a *fixed-parameter tractable* preprocessing algorithm, parameterized by the size of the head. There is an algorithm that, given a graph G and integer k such that G contains an antler decomposition (A, H, R) with $|H| = k$, runs in time $2^{\mathcal{O}(k^5)} \cdot n^{\mathcal{O}(1)}$ and outputs a set of at least k vertices that belong to an optimal feedback vertex set. For each fixed value of k , this yields a preprocessing algorithm to detect vertices that belong to an optimal solution if there is a simple certificate of their membership in an optimal solution.

In fact, Donkers and Jansen gave a more general algorithm; this is where z -antlers for $z > 1$ make an appearance. Recall that for a 1-antler decomposition (A, H, R) of a graph G , the graph $G[A \cup H]$ must contain a collection \mathcal{C} of $|H|$ vertex-disjoint cycles. These cycles certify that the set H is an optimal feedback vertex set in the graph $G[A \cup H]$. In fact, the feedback vertex set H in $G[A \cup H]$ is already optimal for the subgraph $\mathcal{C} \subseteq G[A \cup H]$, and that subgraph \mathcal{C} is structurally simple because each of its connected components (which is a cycle) has a feedback vertex set of size $z = 1$. This motivates the following definition of a z -antler decomposition for $z > 1$: the set H should be an optimal feedback vertex set

for the subgraph $G[A \cup H]$, and moreover, there should be a subgraph $\mathcal{C}_z \subseteq G[A \cup H]$ such that (1) H is an optimal feedback vertex set in \mathcal{C}_z , and (2) each connected component of \mathcal{C}_z has a feedback vertex set of size at most z . So for a z -antler decomposition (A, H, R) of a graph G , there is a certificate that H is part of an optimal solution in the overall graph G that consists of the decomposition together with the subgraph $\mathcal{C}_z \subseteq G[A \cup H]$ for which H is an optimal solution. The complexity of verifying this certificate scales with z : it comes down to verifying that $H \cap V(C)$ is indeed an optimal feedback vertex set of size at most z for each connected component of the subgraph \mathcal{C}_z . Donkers and Jansen presented an algorithm that, given integers $k \geq z \geq 0$ and a graph G that contains a z -antler decomposition whose head has size k , outputs a set of at least k vertices that belongs to an optimal feedback vertex set in time $2^{\mathcal{O}(k^5 z^2)} n^{\mathcal{O}(z)}$. For each fixed choice of k and z , this gives a reduction rule (that can potentially be applied numerous times on an instance) to reduce the search space if the preconditions are met.

Our contribution. We investigate search-space reduction for ODD CYCLE TRANSVERSAL, thereby continuing the line of research proposed by Donkers and Jansen [4]. We introduce the notion of *tight odd cycle cuts* to provide efficiently verifiable witnesses that a certain vertex set belongs to an optimal odd cycle transversal, and present algorithms to find vertices that belong to an optimal solution in inputs that admit such witnesses.

To be able to state our main result, we introduce the corresponding terminology. An *odd cycle cut* (OCC) in an undirected graph G is a partition of its vertex set into three parts: the bipartite part B (which is required to induce a bipartite subgraph of G), the cut part C (which is required to contain all neighbors of B), and the rest $R = V(G) \setminus (B \cup C)$. An odd cycle cut is called *tight* if the set C forms an optimal odd cycle transversal for the graph $G[B \cup C]$. In this case, it is easy to see that there is an optimal odd cycle transversal in G that contains all vertices of C , since all odd cycles through B are intersected by C . A tight OCC (B, C, R) has *order* z if there is a subgraph \mathcal{C}_z of $G[B \cup C]$ for which C is an optimal odd cycle transversal, and for which each connected component of \mathcal{C}_z has an odd cycle transversal of size at most z . This means that for $z = 1$, if there is such a subgraph $\mathcal{C}_z \subseteq G[B \cup C]$, then there is one consisting of $|C|$ vertex-disjoint odd cycles. We use the term z -tight OCC to refer to a tight OCC of order z . Our notion of z -tight OCCs forms an analogue of z -antler decompositions. Note that the requirement that C contains *all* neighbors of B is slightly more restrictive than in the FEEDBACK VERTEX SET case. We need this restriction for technical reasons, but discuss potential relaxations in Section 7.

Similarly to the setting of z -antlers for FEEDBACK VERTEX SET, assuming $P \neq NP$ there is no polynomial-time algorithm that always finds a tight OCC in a graph if one exists; not even in the case $z = 1$. We therefore develop algorithms that are efficient for small k and z . The following theorem captures our main result, which is an OCT-analogue of the antler-based preprocessing algorithm for FVS. The *width* of an OCC (B, C, R) is defined as $|C|$. Our theorem shows that for constant z we can efficiently find k vertices that belong to an optimal solution, if there is a z -tight OCC of width k .

► **Theorem 1.** *There is a deterministic algorithm that, given a graph G and integers $k \geq z \geq 0$, runs in $2^{\mathcal{O}(k^{33} z^2)} \cdot n^{\mathcal{O}(z)}$ time and either outputs at least k vertices that belong to an optimal solution for ODD CYCLE TRANSVERSAL, or concludes that G does not contain a z -tight OCC of width k .*

One may wonder whether it is feasible to have more control over the output, by having the algorithm output a z -tight OCC (B, C, R) of width k , if one exists. However, a small adaptation of a $W[1]$ -hardness proof for antlers [4, Theorem 3.7] shows that the corresponding algorithmic task is $W[1]$ -hard even for $z = 1$. This explains why the algorithm outputs a vertex set that belongs to an optimal solution, rather than a z -tight OCC.

In terms of techniques, our algorithm combines insights from the previous work on antlers [4] with ideas in the representative-set based kernelization [15] for ODD CYCLE TRANSVERSAL. The global idea behind the algorithm is to repeatedly simplify the graph, while preserving the structure of z -tight OCCs, to arrive at the following favorable situation: if there was a z -tight OCC of width k in the input, then the reduced graph has a z -tight OCC (B, C, R) of the same width that satisfies $|B| \in k^{\mathcal{O}(1)}$. At that point, we can use color coding with a set of $k^{\mathcal{O}(1)}$ colors to ensure that the structure $B \cup C$ gets colored in a way that makes it tractable to identify it. The simplification steps on the graph are inspired by the kernelization for ODD CYCLE TRANSVERSAL and involve the computation of a *cut covering set* of size $k^{\mathcal{O}(1)}$ that contains a minimum three-way $\{X, Y, Z\}$ -separator for all possible choices of sets $\{X, Y, Z\}$ drawn from a terminal set T of size $k^{\mathcal{O}(1)}$. The existence of such sets follows from the matroid-based tools of Kratsch and Wahlström [15]. We can avoid the randomization incurred by their polynomial-time algorithm by computing a cut covering set in $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ time deterministically. Compared to the kernelization for ODD CYCLE TRANSVERSAL, a significant additional challenge we face in this setting is that the size of OCTs in the graph can be arbitrarily large in terms of the parameter k . Our algorithm is looking for a *small* region of the graph in which a vertex set exists with a simple certificate for its membership in an optimal solution; it cannot afford to learn the structure of global OCTs in the graph. This local perspective poses a challenge when repeatedly simplifying the graph: we not only have to be careful how these operations affect the total solution size in G , but also how these modifications affect the existence of simple certificates for membership in an optimal solution. This is why our reduction step works with three-way separators, rather than the two-way separators that suffice to solve or kernelize OCT.

Organization. The remainder of this work is organized as follows. The first twelve pages of the manuscript present the key statements and ideas. For statements marked (\star), the proof can be found in the full version [12]. After presenting preliminaries on graphs in Section 2, we define (tight) OCCs in Section 3 and explore some of their properties. In Section 4 we show how color coding can be used to find an OCC whose bipartite part is connected and significantly larger than its cut. Given such an OCC, we show in Section 5 how to simplify the graph while preserving the essential structure of odd cycles in the graph. This leads to an algorithm that finds vertices belonging to an optimal solution the presence of a tight OCC in Section 6. Finally, we conclude in Section 7.

2 Preliminaries

Graphs. We only consider finite, undirected, simple graphs. Such a graph G consists of a set $V(G)$ of vertices and a set $E(G) \subseteq \binom{V(G)}{2}$ of edges. For ease of notation, we write uv for an undirected edge $\{u, v\} \in E(G)$; note that $uv = vu$. When it is clear which graph is referenced from context, we write n and m to denote the number of vertices and edges in this graph respectively. For a vertex $v \in V(G)$, its open neighborhood is $N_G(v) := \{u \in V(G) \mid uv \in E(G)\}$ and its closed neighborhood is $N_G[v] := N_G(v) \cup \{v\}$. For a vertex set $S \subseteq V(G)$ we define its open neighborhood as $N_G(S) := (\bigcup_{v \in S} N_G(v)) \setminus S$

and its closed neighborhood as $N_G[S] := \bigcup_{v \in S} N_G[v]$. The subgraph of G induced by a vertex set $S \subseteq V(G)$ is the graph $G[S]$ on vertex set S with edges $\{uv \in E(G) \mid \{u, v\} \subseteq S\}$. We use $G - S$ as a shorthand for $G[V(G) \setminus S]$ and write $G - v$ instead of $G - \{v\}$ for singletons. A *walk* is a sequence of (not necessarily distinct) vertices (v_1, \dots, v_k) such that $v_i, v_{i+1} \in E(G)$ for each $i \in [k - 1]$. The walk is *closed* if we additionally have $v_k, v_1 \in E(G)$. A *cycle* is a closed walk whose vertices are all distinct. The *length* of a cycle (v_1, \dots, v_k) is k . A *path* is a walk whose vertices are all distinct. The *length* of a path (v_1, \dots, v_k) is $k - 1$. The vertices v_1, v_k are the *endpoints* of the path. For two (not necessarily disjoint) vertex sets S, T of a graph G , we say that a path $P = (v_1, \dots, v_k)$ in G is an (S, T) -path if $v_1 \in S$ and $v_k \in T$. If one (or both) of S and T contains only one element, we may write this single element instead of the singleton set consisting of it.

The *parity* of a path or cycle refers to the parity of its length. For a walk $W = (v_1, \dots, v_k)$, we refer to its vertex set as $V(W) = \{v_1, \dots, v_k\}$. Observe that if W is a closed walk of odd parity (a *closed odd walk*), then the graph $G[V(W)]$ contains a cycle of odd length (an *odd cycle*): any edge connecting two vertices of $V(W)$ that are not consecutive on W splits the walk into two closed subwalks, one of which has odd length.

For a positive integer q , a *proper q -coloring* of a graph G is a function $f: V(G) \rightarrow \{0, \dots, q - 1\}$ such that $f(u) \neq f(v)$ for all $uv \in E(G)$. A graph G is *bipartite* if its vertex set can be partitioned into two *partite sets* $L \dot{\cup} R$ such that no edge has both of its endpoints in the same partite set. It is well-known that the following three conditions are equivalent for any graph G : (1) G is bipartite, (2) G admits a proper 2-coloring, and (3) there is no cycle of odd length in G . An *odd cycle transversal* (OCT) of a graph G is a set $S \subseteq V(G)$ such that $G - S$ is bipartite. An *independent set* is a vertex set S such that $G[S]$ is edgeless. We say that a vertex set X in a graph G *separates* two (not necessarily) disjoint vertex sets S and T if no connected component of $G - X$ simultaneously contains a vertex from S and a vertex from T . For a collection $\{T_1, \dots, T_m\}$ of (not necessarily disjoint) vertex sets in a graph G , we say that a vertex set X is an $\{T_1, \dots, T_m\}$ -separator if X separates all pairs (T_i, T_j) for $i \neq j$. Note that X is allowed to intersect $\bigcup_{i \in [m]} T_i$.

The following lemma gives a simple sufficient condition for a graph to be bipartite.

► **Lemma 2.** *Let G be a graph and let $V_L \cup V_0 \cup V_R = V(G)$ be a partition of its vertices such that V_0 is a $\{V_L, V_R\}$ -separator. If there exist proper 2-colorings $f_L: (V_0 \cup V_L) \rightarrow \{0, 1\}$ and $f_R: (V_0 \cup V_R) \rightarrow \{0, 1\}$ of $G[V_0 \cup V_L]$ and $G[V_0 \cup V_R]$ respectively such that $f_L(v_0) = f_R(v_0)$ for every $v_0 \in V_0$, then G is bipartite.*

Proof. To show that G is bipartite, we provide a proper 2-coloring of the graph. We define this coloring $f: V(G) \rightarrow \{0, 1\}$ such that $f(v_0) = f_L(v_0) (= f_R(v_0))$ for every $v_0 \in V_0$, $f(v_L) = f_L(v_L)$ for every $v_L \in V_L$ and $f(v_R) = f_R(v_R)$ for every $v_R \in V_R$. To see that f is a proper 2-coloring, we show that no edge $e \in E(G)$ is monochromatic under f .

By the assumption that V_0 is a separator, each edge $e \in E(G)$ is contained in $G[V_0 \cup V_L]$ or $G[V_0 \cup V_R]$ (or both). If e is an edge in the former, its endpoints are colored the same as in f_L and are therefore bichromatic. The analogous argument for f_R holds when e is an edge of the latter. ◀

The next lemma captures the main idea behind the iterative compression algorithm [19] (cf. [3, §4.4]) for solving ODD CYCLE TRANSVERSAL. Given a (potentially suboptimal) odd cycle transversal W of a graph, it shows that the task of finding an odd cycle transversal disjoint from W whose removal leaves a bipartite graph with $W_0, W_1 \subseteq W$ in opposite partite sets of its bipartition is equivalent to separating two vertex sets derived from a baseline bipartition of $G - W$. Our statement below is implied by Claim 1 in the work of Jansen and de Kroon [9].

► **Lemma 3** ([9, Claim 1]). *Let W be an OCT in graph G . For each partition of $W = W_0 \cup W_1$ into two independent sets, for each proper 2-coloring c of $G - W$, we have the following equivalence for each $X \subseteq V(G) \setminus W$: the graph $G - X$ has a proper 2-coloring with W_0 color 0 and W_1 color 1 if and only if the set X separates A from R in the graph $G - W$, with:*

$$A = (N_G(W_0) \cap c^{-1}(0)) \cup (N_G(W_1) \cap c^{-1}(1)),$$

$$R = (N_G(W_0) \cap c^{-1}(1)) \cup (N_G(W_1) \cap c^{-1}(0)).$$

Multiway cuts. Let $\mathcal{T} = (T_1, \dots, T_s)$ be a partition of a set $T \subseteq V(G)$ of *terminal* vertices in an undirected graph G . A *multiway cut* of \mathcal{T} in G is a vertex set $X \subseteq V(G)$ such that for each pair $t_i, t_j \in T \setminus X$ that belong to different parts of partition \mathcal{T} , the graph $G - X$ does not contain a path from t_i to t_j . A *restricted multiway cut* of \mathcal{T} is a vertex set X that is a multiway cut for \mathcal{T} such that $X \cap T = \emptyset$, i.e., it does not contain any terminals.

For a positive integer s , a *generalized s -partition* of a set T is a partition $\mathcal{T}^* = (T_0, T_1, \dots, T_s, T_X)$ of T into $s + 2$ parts, some of which can be empty. The parts T_0 and T_X play a special role, which are the *free* and *deleted* part of \mathcal{T}^* , respectively. Let $T' = T_1 \cup \dots \cup T_s$. A *multiway cut* of \mathcal{T}^* is a (non-restricted) multiway cut in $G - T_X$ of the partition $\mathcal{T} = (T_1, \dots, T_s)$ of T' . Hence the vertices of T_X are deleted from the graph, while no cut constraints are imposed on the vertices of T_0 .

A *minimum multiway cut* of a generalized s -partition \mathcal{T}^* in a graph G is a minimum-cardinality vertex set that satisfies the requirements of a multiway cut for \mathcal{T}^* . The following cut covering lemma by Kratsch and Wahlström will be useful for our algorithm.

► **Theorem 4** ([15, Theorem 5.14]). *Let G be an undirected graph on n vertices with a set $T \subseteq V(G)$ of terminal vertices, and let $s \in \mathbb{N}$ be a constant. There is a set $Z \subseteq V(G)$ with $|Z| = \mathcal{O}(|T|^{s+1})$ such that Z contains a minimum multiway cut of every generalized s -partition \mathcal{T}^* of T , and we can compute such a set in randomized polynomial time with failure probability $\mathcal{O}(2^{-n})$.*

For a generalized s -partition $\mathcal{T} = (T_0, T_1, \dots, T_s, T_X)$ of a terminal set $T \subseteq V(G)$ in an undirected graph G , we call a multiway cut X of \mathcal{T} *restricted* if it satisfies $X \cap (\bigcup_{i=1}^s T_i) = \emptyset$. Hence a restricted multiway cut does not delete any vertex that is active as a terminal in the generalized partition. A minimum *restricted* multiway cut of \mathcal{T} is a restricted multiway cut whose size is minimum among all restricted multiway cuts.

The following lemma shows that the randomization in the polynomial-time algorithm by Kratsch and Wahlström can be avoided by the use of a single-exponential FPT algorithm, and that the cut covering set can be adapted to work for *restricted* multiway cuts as long as we have a bound on their size.

► **Lemma 5** (★). *Let $s \in \mathbb{N}$ be a constant. There is a deterministic algorithm that, given an undirected n -vertex graph G and a set $T \subseteq V(G)$ of terminals, runs in time $2^{\mathcal{O}(|T|)} \cdot n^{\mathcal{O}(1)}$ and computes a set $Z \subseteq V(G)$ with $|Z| = \mathcal{O}(|T|^{2s+2})$ with the following guarantee: for each generalized s -partition \mathcal{T} of T , if there is a restricted multiway cut for \mathcal{T} of size at most $|T|$ in G , then the set Z contains a minimum restricted multiway cut of \mathcal{T} .*

3 Odd Cycle Cuts

In order to extend the “antler” framework of [4] to ODD CYCLE TRANSVERSAL (OCT), we define a problem-specific decomposition which we term *Odd Cycle Cuts* (OCCs). Our decompositions have three parts – a bipartite induced subgraph X_B , a vertex separator X_C (which we call the *head*), and a remainder X_R .

► **Definition 6** (Odd Cycle Cut). *Given a graph G , a partition (X_B, X_C, X_R) of $V(G)$ is an Odd Cycle Cut (OCC) if (1) $G[X_B]$ is bipartite, (2) there are no edges between X_B and X_R , and (3) $X_C \cup X_B \neq \emptyset$.*

We say $|X_C|$ is the *width* of an OCC, and observe that X_C hits all odd cycles in $G - X_R$. We denote the minimum size of an OCT in G by $\text{oct}(G)$.

► **Observation 7.** *If (X_B, X_C, X_R) is an OCC in G , then $|X_C| \geq \text{oct}(G[X_C \cup X_B])$.*

Analogous to z -antlers [4], here we define a *tight OCC* as a special case of an OCC. For a graph G , a set $X_C \subseteq V(G)$ and an integer z , an X_C -*certificate* of order z is a subgraph H of G such that X_C is an optimal OCT of H , and for each component H' of H we have $|X_C \cap V(H')| \leq z$. Throughout the paper, and starting with the following definition, we will use the convention of referring to a tight OCC as (A_B, A_C, A_R) to emphasize its stronger guarantees compared to an arbitrary OCC (X_B, X_C, X_R) .

► **Definition 8** ((z) -tight OCC). *An OCC (A_B, A_C, A_R) of a graph G is tight when $|A_C| = \text{oct}(G[A_C \cup A_B])$. Furthermore, (A_B, A_C, A_R) is a tight OCC of order z (equivalently, z -tight OCC) if $G[A_C \cup A_B]$ contains an A_C -certificate of order z .*

Note this definition naturally implies $\text{oct}(G) = |A_C| + \text{oct}(G[A_R])$: the union of A_C with a minimum OCT in $G[A_R]$ forms an OCT for G (since A_C separates A_B from A_R) for which the requirement $|A_C| = \text{oct}(G[A_C \cup A_B])$ guarantees optimality. The main result of this section is that assuming a graph G has a z -tight OCC, there exists a z -tight OCC (A_B, A_C, A_R) such that the number of components in $G[A_B]$ is bounded in terms of z and $|A_C|$. This is an extension of [4, Lemma 4.6], and we defer its proof to the full version of this paper [12].

► **Lemma 9** (★). *Let (A_B, A_C, A_R) be a z -tight OCC in a graph G for some $z \geq 0$. There exists a set $A'_B \subseteq A_B$ such that $(A'_B, A_C, A_R \cup A_B \setminus A'_B)$ is a z -tight OCC in G and $G[A'_B]$ has at most $z^2|A_C|$ components.*

Finally, we introduce the notion of an *imposed separation problem* whose solutions naturally correspond to odd cycle transversals of specific subgraphs.

► **Definition 10.** *Let (X_B, X_C, X_R) be an OCC of G , and let $f_B: X_B \rightarrow \{0, 1\}$ be a proper 2-coloring of $G[X_B]$. Let $C_1, C_2 \subseteq X_C$ be two disjoint subsets of X_C and let $f_C: C_1 \rightarrow \{0, 1\}$ be a (not necessarily proper) 2-coloring of the vertices in C_1 . Based on this 4-tuple of objects (C_1, C_2, f_C, f_B) , we define three (potentially overlapping) subsets $A, R, N \subseteq X_B$.*

1. *Let A be the set of vertices $v_b \in X_B$ with a neighbor $v_c \in C_1$ such that $f_B(v_b) = f_C(v_c)$.*
2. *Let R be the set of vertices $v_b \in X_B$ with a neighbor $v_c \in C_1$ such that $f_B(v_b) \neq f_C(v_c)$.*
3. *Finally, let $N := N_G(C_2) \cap X_B$.*

We refer to the problem of finding a smallest $\{A, R, N\}$ -separator in $G[X_B]$ as the $\{A, R, N\}$ -separation problem imposed onto $G[X_B]$ by (C_1, C_2, f_C, f_B) .

To see the connection between solutions and OCTs, one may let C_1 and f_B in this definition correspond to W and c respectively in Lemma 3, while the color classes of f_C correspond to the sets W_0 and W_1 respectively. As shown below in Lemma 11, we can recognize parts of tight OCCs as optimal solutions to specific imposed separation problems.

Although Definition 10 requires f_B and f_C to be colorings of X_B and C_1 respectively, we sometimes abuse the notation by providing colorings whose domains are supersets of these intended domains. In these cases, one may interpret the definition of the imposed separation problem as if given the restrictions of these colorings to their respective intended domains.

One important role of these separation problems is to allow us to characterize intersections of two OCCs when at least one is tight. Specifically, in Lemma 11, we show that the intersection of one OCC's head with the other OCC's bipartite part forms an optimal solution to a specific 3-way separation problem, which is even optimal for a corresponding 2-way problem.

► **Lemma 11** (★). *Let (X_B, X_C, X_R) be a (not necessarily tight) OCC in the graph G and let (A_B, A_C, A_R) be a tight OCC in G . Let $f_X: X_B \rightarrow \{0, 1\}$ and $f_A: A_B \rightarrow \{0, 1\}$ be proper 2-colorings of $G[X_B]$ and $G[A_B]$ respectively. Let A , R and N be the three sets to be separated in the separation problem imposed onto $G[X_B]$ by $(X_C \cap A_B, X_C \cap A_R, f_A, f_B)$ and let their names correspond to their roles as defined in Definition 10. Then, $A_C \cap X_B$ is both a minimum-size $\{A, R\}$ -separator and a minimum-size $\{A, R, N\}$ -separator in $G[X_B]$.*

This will prove to be a useful property in Section 5 by which we are able to recognize part of a tight OCC (A_B, A_C, A_R) in an arbitrary graph. We complement it with the statement below, indicating that the intersection $A_C \cap X_B$ is even bounded in size.

► **Lemma 12**. *Let (X_B, X_C, X_R) be a (not necessarily tight) OCC in the graph G and let (A_B, A_C, A_R) be a tight OCC in G . Then $|A_C \cap X_B| \leq |X_C|$.*

Proof. Suppose for contradiction that $|A_C \cap X_B| > |S|$. Then, $A'_C := (A_C \setminus X_B) \cup (X_C \cap (A_B \cup A_C))$ is a subset of $A_B \cup A_C$ that is strictly smaller than A_C . Now, showing that A'_C is an OCT of $G[A_B \cup A_C]$ contradicts the assumption that A_C is a smallest such OCT by virtue of (A_B, A_C, A_R) being a tight OCC.

To show that A'_C is an OCT of $G[A_B \cup A_C]$, we let F be an arbitrary odd cycle in this graph and show that it intersects A'_C . First, if F intersects X_C , it intersects A'_C in particular, since $X_C \cap (A_B \cup A_C) \subseteq A'_C$.

Otherwise, since X_C separates X_B and X_R in G , F is completely contained in either $G[X_B]$ or $G[X_R]$. The former is not possible, since $G[X_B]$ is bipartite by assumption, so F lives in $G[X_R]$. Furthermore, since F was assumed to live in $G[A_B \cup A_C]$ and $G[A_B]$ is bipartite, F intersects A_C . In particular, as we found F to live in $G[X_R]$, it intersects $A_C \cap X_R$ which is a subset of A'_C by construction. Hence, F intersects A'_C in any case. ◀

4 Finding Odd Cycle Cuts

Our ultimate goal is to show that if the graph contains any tight OCC (X_B, X_C, X_R) with $|X_C| \leq k$, then we can produce a tight OCC with $|X_C| \leq k$ and $|X_B|$ upper-bounded by some function of k . To achieve this, we first show that we can efficiently find some OCC where $|X_B|$ is large enough, and then (in Section 5) that we can reduce any such cut so that $|X_B|$ is small without destroying any essential structure of the input graph.

Specifically, we say an OCC (X_B, X_C, X_R) is *reducible* with respect to some function g_r if $|X_B| > g_r(|X_C|)$. Our results all hold for a specific polynomial $g_r(x)$ in $\Theta(x^{16})$. Its definition relies on Lemma 5 in which sets Z and T are specified. Setting the value of s in this lemma to 3 yields the existence of a constant $c \in \mathbb{N}$ such that $|Z| \leq c \cdot |T|^8$ for large enough $|T|$. Given this constant c , we define $g_r: \mathbb{N} \rightarrow \mathbb{N}$ as $g_r(x) = (6(2^8c + 1)^2 + 2^8c) \cdot x^{16}$.

We say an OCC (X_B, X_C, X_R) is a *single-component* OCC if $G[X_B]$ is connected. Given a graph G , our goal is to output a reducible OCC efficiently assuming that G contains a single-component OCC (X_B, X_C, X_R) with $|X_B| > g_r(2|X_C|)$ and $|X_C| \leq k$. We achieve this by color coding of the vertices in G (see the full version [12] for details). Consider a coloring $\chi: V(G) \rightarrow \{\check{B}, \check{C}\}$. For an integer ℓ , an OCC (X_B, X_C, X_R) with $|X_B| \geq \ell$ is *ℓ -property*

colored by χ if $X_C \subseteq \chi^{-1}(\dot{C})$ and there is a set of ℓ vertices of X_B that are colored \dot{B} and induce a connected subgraph of G . First, we show how to construct an OCC with large X_B from a proper coloring.

► **Lemma 13 (★).** *Given a graph G , integers k, ℓ , and a coloring $\chi: V(G) \rightarrow \{\dot{B}, \dot{C}\}$ of $V(G)$ that ℓ -properly colors a single-component OCC (X_B, X_C, X_R) with $|X_C| \leq k$, an OCC (X'_B, X'_C, X'_R) such that $|X'_B| \geq \ell$ and $|X'_C| \leq 2k$ can be found in polynomial time.*

Proof sketch. We iterate over the connected components of $G[\chi^{-1}(\dot{B})]$. For any component which is both large ($\geq \ell$) and bipartite, we try to find an OCC of small enough width where the component is contained in the X_B side of the cut. To do this, we use the machinery of bipartite separations introduced in Jansen et al. [11] (see the full version [12] for details). Intuitively, given a vertex set C which induces a connected bipartite subgraph, they either find a set of at most $2k$ vertices which separates $C' \supseteq C$ from the remainder of the graph so that $G[C']$ is bipartite, or certify that C is not part of X_B for any OCC with width $\leq k$. ◀

Now, we use this coloring scheme to find a reducible OCC, assuming that a graph G has a single-component OCC (X_B, X_C, X_R) with large X_B .

► **Lemma 14.** *There exists a $2^{\mathcal{O}(k^{16})}n^{\mathcal{O}(1)}$ -time algorithm that, given a graph G and an integer k , either determines that G does not contain a single-component OCC (X_B, X_C, X_R) of width at most k with $|X_B| > g_r(2k)$ or outputs a reducible OCC in G .*

Proof. We will invoke the algorithm from Lemma 13 multiple times for $\ell = g_r(2k) + 1$. If we supply a coloring that ℓ -properly colors (X_B, X_C, X_R) , then the algorithm is guaranteed to find an OCC (X'_B, X'_C, X'_R) such that $|X'_B| > g_r(2k)$ and $|X'_C| \leq 2k$, which is reducible as $|X'_B| > g_r(2k) \geq g_r(|X'_C|)$. If all relevant colorings fail to find such a reducible OCC, then we can conclude that G does not contain a single-component OCC (X_B, X_C, X_R) with $|X_C| \leq k$ and $|X_B| \geq \ell > g_r(2k)$.

Let $X'_B \subseteq X_B$ be an arbitrary vertex set of size ℓ that induces a connected subgraph of G . Since $G[X_B]$ is connected, such X'_B must exist. Observe that we obtain an ℓ -proper coloring if $X_C \cup X'_B$ are colored correctly. Let $s = |X_C \cup X'_B| = k + g_r(2k) + 1 = \mathcal{O}(k^{16})$.

Using an (n, k) -universal set, which is a well-known pseudorandom object [17, 3] used to derandomize applications of color coding (see [3, Theorem 5.20]), we can construct a family of $2^{\mathcal{O}(s)} \log n$ many subsets $A_1, \dots, A_{2^{\mathcal{O}(s)} \log n}$ with the guarantee that for each set $S \subseteq V(G)$ of size s , for each subset S' of S , there exists a set in the family with $A_i \cap S = S'$. This can be done in $2^{\mathcal{O}(s)} n \log n = 2^{\mathcal{O}(k^{16})} n \log n$ time. From this family, we can construct a family of colorings that is guaranteed to include one that ℓ -properly colors a suitable OCC (X_B, X_C, X_R) if one exists. To derive a coloring χ_i from a member $A_i \subseteq V(G)$ of the (n, s) -universal set, it suffices to pick $\chi(a \in A) = \dot{R}$ and $\chi(a \notin A) = \dot{B}$.

We run the $n^{\mathcal{O}(1)}$ -time algorithm from Lemma 13 for each coloring, which results in the overall runtime $2^{\mathcal{O}(k^{16})} n^{\mathcal{O}(1)}$. ◀

5 Reducing Odd Cycle Cuts

Given an OCC (X_B, X_C, X_R) of G with $|X_B| > g_r(|X_C|)$, the next step is to “shrink” X_B in a way that preserves some of the structure of the input graph. In this section, we give a reduction to do this and prove that it preserves the general structure of minimum-size OCTs and of tight OCCs in the graph. The reduction starts with a marking scheme that is discussed separately in Section 5.1. We give the full reduction, which includes this marking scheme as a subroutine, in Section 5.2. The reduction will only affect $G[X_B]$ and the edge set between X_B and X_C , which already ensures that an important part of the input graph is maintained.

5.1 A marking scheme for the reduction

The goal of the marking scheme is to mark a set $B^* \subseteq X_B$ of size $|X_C|^{\mathcal{O}(1)}$ as “interesting” vertices that the reduction should not remove or modify. Intuitively, we want this set to contain vertices which we expect might be part of the cut part of a tight OCC in G . More precisely, we guarantee that for every tight OCC in G there is a (possibly different) tight OCC (A_B, A_C, A_R) such that $A_C \cap X_B$ is contained in the marked set B^* .

As seen in Lemma 11, for every tight OCC (A_B, A_C, A_R) in the graph, the intersection $A_C \cap X_B$ forms an optimal solution to a specific imposed separation problem (Definition 10). As such, it suffices if B^* is a *cut covering set* for these imposed separation problems.

Indeed, the key ingredient of the algorithm presented below is the computation of such a cut covering set. Preceding this computation is a graph reduction ensuring that the computed set covers precisely the imposed separation problems. In Lemma 5, we will show that a cut covering set can be computed in deterministic FPT time parameterized by the size of the terminal set, which leads to a total running time of $2^{\mathcal{O}(|X_C|)} n^{\mathcal{O}(1)}$ time for the marking step.

► **Marking step.** *Consider the following algorithm.*

Input: A graph G and an OCC (X_B, X_C, X_R) of G .

Output: Marked vertices $B^* \subseteq X_B$.

1. Find a proper 2-coloring $f_X: X_B \rightarrow \{0, 1\}$ of $G[X_B]$.
2. Construct an auxiliary (undirected) graph G' , initialized to a copy of $G[X_B]$. For each $v \in X_C$, do the following:
 - Add vertices $v^{(0)}$ and $v^{(1)}$ to G' .
 - For each neighbor $u \in N_G(v) \cap X_B$, add an edge $v^{(f_X(u))}u$.
 Let T be the set $\{v^{(i)} \mid v \in X_C, i \in \{0, 1\}\}$. Note that $|T| = 2|X_C|$.
3. Compute a cut covering set $B^* \subseteq V(G')$ via Lemma 5 such that for every partition $\mathcal{T}^* = (T_0, T_1, T_2, T_3, T_X)$ of T , the set B^* contains a minimum-size solution to the following problem:
 - find a vertex set $S \subseteq V(G') \setminus (T_1 \cup T_2 \cup T_3)$ such that S separates T_i and T_j in the graph $G' - T_X$ for all $1 \leq i < j \leq 3$,
as long as this problem has a solution of size at most $|T|$.

► **Lemma 15 (★).** *Let B^* be constructed as in the Marking step when given the graph G and an OCC (X_B, X_C, X_R) of G as input. If there exists a z -tight OCC (A_B, A_C, A_R) in G , then there exists a z -tight OCC (A_B^*, A_C^*, A_R^*) in G with $|A_C^*| = |A_C|$ and with $A_C^* \cap X_B \subseteq B^*$.*

Proof sketch. Let $f_X: X_B \rightarrow \{0, 1\}$ be the 2-coloring obtained in step 1 of the Marking step, let $f_A: A_B \rightarrow \{0, 1\}$ be a proper 2-coloring of $G[A_B]$ and consider the separation problem imposed onto $G[X_B]$ by $(X_C \cap A_B, X_C \cap A_R, f_A, f_B)$. Let A , R and N be the three sets to be separated in this problem with their names corresponding to their roles as in Definition 10.

By putting the correct copy of each vertex from $A_B \cap X_C$ into T_1 and T_2 respectively, putting both copies of vertices from $A_R \cap X_C$ into T_3 and putting both copies of vertices from $A_C \cap X_C$ into T_X , we obtain a partition $(\emptyset, T_1, T_2, T_3, T_X)$ of the set T defined in step 2, such that the corresponding separation problem has the same solution space as the $\{A, R, N\}$ -separation problem imposed onto $G[X_B]$. By construction of B^* in step 3, there is a set $S \subseteq B^*$ (possibly different from $A_C \cap X_B$) that is an optimal $\{A, R, N\}$ -separator in $G[X_B]$. To construct the tight OCC (A_B^*, A_C^*, A_R^*) , we use this set S as replacement for $A_C \cap X_B$, which is also a minimum-size $\{A, R, N\}$ -separator in $G[X_B]$ by Lemma 11.

15:12 Preprocessing to Reduce the Search Space for Odd Cycle Transversal

As such, we define $A_C^* := (A_C \setminus X_B) \cup S$. To define A_B^* , let U be the set of vertices from $X_B \setminus S$ that are not reachable from N in $G[X_B] - S$. Now, we define $A_B^* := (A_B \setminus X_B) \cup U$. Finally, we define $A_R^* := V(G) \setminus (A_B^* \cup A_C^*)$. Clearly, this 3-partition of $V(G)$ satisfies the constraints $|A_C^*| = |A_C|$ and $A_C^* \cap X_B \subseteq B^*$. We proceed by showing that it satisfies the three additional properties required to be a z -tight OCC.

First, to see that $G[A_B^*]$ is bipartite, we note that A_B^* only contains vertices from A_B and $X_B \setminus S$. Both are vertex sets that induce a bipartite subgraph. Then, noting the correspondence between the sets A and R obtained from the separation problem and the sets A and R as in Lemma 3, we invoke this lemma on $G[(X_C \cap A_B) \cup X_B]$ with $c = f_A$ and with W_0 and W_1 being the two color classes of this coloring restricted to $X_C \cap A_B$. It follows that the vertices from A_B^* in $X_B \setminus S$ can be properly 2-colored by a coloring f that agrees with f_A on the vertex set $X_C \cap A_B$ that separates $A_B^* \cap X_R$ and $A_B^* \cap X_B$. As these two vertex sets are properly colored by f_A and f respectively, these colorings combine to a proper 2-coloring of the entire graph $G[A_B^*]$ (see Lemma 2).

Secondly, a case distinction shows that there are no edges between A_B^* and A_R^* . It combines the fact that $A_C \cap X_B$ is an $\{A, R, N\}$ -separator in $G[X_B]$ – thereby in particular separating $A_B \cap X_B$ from N in $G[X_B]$ – and the fact that A_B^* only contains vertices that already belonged to A_B and vertices from X_B that are not reachable from N in $G[X_B] - A_C^*$.

Finally, it remains to show that (A_B^*, A_C^*, A_R^*) has an A_C^* -certificate of order z . To prove this, we show that the order- z certificate D of the original OCC (A_B, A_C, A_R) is also an order- z certificate in (A_B^*, A_C^*, A_R^*) . The main effort here is to prove that D even lives in $A_B^* \cup A_C^*$, after which it is easy to see that it is also an order- z certificate for our new OCC.

As Lemma 11 guarantees that $A_C \cap X_B$ is not only an optimal $\{A, R, N\}$ -separator in $G[X_B]$ but even an optimal $\{A, R\}$ -separator in this graph, it contains exactly one vertex from every path of a maximum packing \mathcal{P} of pairwise vertex-disjoint (A, R) -paths in $G[X_B]$, due to Menger's theorem [20, Theorem 9.1]. Likewise, $A_C^* \cap X_B = S$ is also an optimal $\{A, R\}$ -separator in $G[X_B]$ and hence also contains exactly one vertex from every path of \mathcal{P} .

Intuitively, for any path $P \in \mathcal{P}$, a vertex on this path that stops being reachable from one endpoint of P when sliding the picked vertex along the path, starts becoming reachable from the other endpoint of P . As both endpoints of P belong to $A \cup R$ and S only differs from $A_C \cap X_B$ by which vertex is picked from each path in \mathcal{P} , it cannot drastically alter which vertices are reachable from $A \cup R$, which in turn are all vertices that end up in A_B^* .

Using the observation that A_B and A_B^* are separated from N by A_C and A_C^* respectively, we see that all vertices that are disconnected from $A \cup R$ by substituting $A_C \cap X_B$ for S are in particular also disconnected from N . Thereby, these vertices end up in A_B^* . This shows that $(A_B \cup A_C) \subseteq (A_B^* \cup A_C^*)$, which implies that the certificate D also lives in the latter. ◀

5.2 Simplifying the graph

Our eventual reduction starts with the Marking step from the previous section, after which the graph is modified in a way that leaves marked vertices untouched. We want the reduction to preserve the general structure of optimal OCTs and tight OCCs in the input graph. As this is governed by the locations and interactions of odd cycles in the graph, we encode this information in a more space-efficient manner using the following reduction.

► **Reduction step.** *Given a graph G and an OCC (X_B, X_C, X_R) of it, we construct a graph G' as follows.*

1. *Use the Marking step with input G and (X_B, X_C, X_R) to obtain the set $B^* \subseteq X_B$.*
2. *Initialize G' as a copy of $G - (X_B \setminus B^*)$.*

3. For every $u, v \in X_C \cup B^*$ and for every parity $p \in \{\text{even}, \text{odd}\}$, check if the subgraph $G[X_B \setminus B^*]$ contains the internal vertex of a (u, v) -path with parity p . If so, then:
- if $p = \text{even}$, add two new vertices x and x' to G and connect both of them to u and v .
 - if $p = \text{odd}$, add four new vertices x, y, x' and y' to G and add the edges $\{u, x\}, \{x, y\}, \{y, v\}, \{u, x'\}, \{x', y'\}$ and $\{y', v\}$.
- Note that we explicitly allow $u = v$ in this step.

Effectively, this reduction deletes the vertices $X_B \setminus B^*$ from the graph. For each pair of neighbors u, v from that set, if the deleted vertices provided an odd (resp. even) path between them, then we insert two vertex-disjoint odd (resp. even) paths between u and v . Hence we shrink the graph while preserving the parity of paths provided by the removed vertices.

As we prove in the full version of this paper [12], the reduction can be performed in $2^{\mathcal{O}(|X_C|)} \cdot n^{\mathcal{O}(1)}$ time and it is guaranteed to output a strictly smaller graph than its input graph whenever it receives an OCC that is reducible with respect to the function g_r as in Section 4. To show that the reduction also preserves OCT and OCC structures, we prove that it satisfies two safety properties formalized below in Lemmas 16 and 17.

► **Lemma 16 (★).** *Let G be a graph, let (X_B, X_C, X_R) be an OCC in G and let G' be the graph obtained by running the Reduction step with these input parameters. For all $z \geq 0$, if there exists a z -tight OCC (A_B, A_C, A_R) in G , then there exists a z -tight OCC (A'_B, A'_C, A'_R) in G' with $|A'_C| = |A_C|$.*

The proof of the lemma above uses Lemma 15 to infer that, for any z -tight OCC (A_B, A_C, A_R) of G , the graph G also contains a z -tight OCC (A_B^*, A_C^*, A_R^*) of the same width such that $A_C^* \subseteq V(G) \cap V(G')$. This allows for the construction of an OCC (A'_B, A'_C, A'_R) in G' with $A'_C = A_C^*$. Then, A'_B can be defined as the union of $A_B^* \cap V(G) \cap V(G')$ and the set of vertices that were added during the reduction to provide a replacement connection between any two vertices from $A_C^* \cup (A_B^* \cap V(G) \cap V(G'))$. Finally, $A'_R := V(G') \setminus (A'_B \cup A'_C)$.

The proof proceeds to show that the resulting partition (A'_B, A'_C, A'_R) is a z -tight OCC of G' . The two main insights used to prove this are the facts that:

- optimal OCTs of G' are disjoint from the set of newly added vertices $V(G') \setminus V(G)$, and
- odd cycles in G can be translated to very similar odd cycles in G' and vice versa.

These insights are also covered in the proof sketch of the second safety property below.

► **Lemma 17 (★).** *Let G be a graph, let (X_B, X_C, X_R) be an OCC in G and let G' be the graph obtained by running the Reduction step with these input parameters. If S' is a minimum-size OCT of G' , then $S' \subseteq V(G) \cap V(G')$ and S' is a minimum-size OCT of G .*

Proof sketch. To see that $S' \subseteq V(G) \cap V(G')$, we show that S' contains none of the newly added vertices in $V(G') \setminus V(G)$. These newly added vertices come in pairs that form degree-2 paths connecting the same endpoints. Consider two such paths and let u and v be the endpoints of both of them. Suppose for contradiction that S' uses an internal vertex p_1 from one path to break an odd cycle F . Then it must also contain an internal vertex p_2 from the other path to break the odd cycle obtained by swapping one path for the other in F . As both these cycles also pass through u and v by construction, substituting p_1 and p_2 for one of u and v in S' yields a strictly smaller solution. This contradicts the optimality of S' .

To see that S' is an OCT of G , suppose for contradiction that $G - S'$ contains an odd cycle F . Every subpath of F that connects two vertices from $V(G) \cap V(G')$ via a path whose internal vertices lie in $G - V(G')$ can be replaced by one of the paths inserted during the construction of G' , with the same endpoints and parity. Substituting every subpath of F that is absent in G' for such a replacement path yields a closed odd walk in $G' - S'$; but this contradicts the fact that S' is an OCT of G' . Hence $S' \subseteq V(G) \cap V(G')$ is an OCT in G .

It remains to show that S' is an OCT of minimum size. Suppose for contradiction that T is a strictly smaller OCT of G . We start by showing how to modify T into an OCT S of G that is at most as large and lives in $V(G) \cap V(G')$. To this end, let $f: V(G) \setminus T \rightarrow \{0, 1\}$ and $f_X: X_B \rightarrow \{0, 1\}$ be proper 2-colorings of $G - T$ and $G[X_B]$ respectively and consider the separation problem imposed onto $G[X_B]$ by $(X_C \setminus T, \emptyset, f, f_B)$. Let A, R, N be the three sets to be separated in this problem with their names corresponding to their roles as in Definition 10. Since the second argument C_2 in the 4-tuple is \emptyset , we obtain $N = \emptyset$.

The fact that $N = \emptyset$ ensures that the separation problem above is merely a 2-way separation problem between the sets A and R in $G[X_B]$. These sets are defined in such a way that, for a suitable choice of input parameters to Lemma 3, they coincide with the sets A and R in this lemma. Applying the lemma in one direction to $G[(X_C \setminus T) \cup X_B]$ with $c = f$ and with W_0 and W_1 being the two color classes of this coloring restricted to $X_C \setminus T$, yields that $T \cap X_B$ is an $\{A, R\}$ -separator in $G[X_B]$. Applying it in the other direction yields that the removal of *any* $\{A, R\}$ -separator T' from G allows for a proper 2-coloring f' of $G[(X_C \setminus T) \cup (X_B \setminus T')]$ that agrees with the coloring f on the vertex set $X_C \setminus T$. As this set separates the subgraphs $G[X_R \setminus T]$ and $G[X_B \setminus T']$ in $G - ((T \setminus X_B) \cup T')$ and these subgraphs are properly 2-colored by f and f' respectively, those two colorings combine to properly 2-color $G - ((T \setminus X_B) \cup T')$ (see Lemma 2). By construction of B^* , there is a minimum-size $\{A, R\}$ -separator T^* in $G[X_B]$ with $T^* \subseteq B^*$. Hence, $S := (T \setminus X_B) \cup T^*$ is an OCT of G that lives in $V(G) \cap V(G')$. Furthermore, since T^* is a minimum-size $\{A, R\}$ -separator in $G[X_B]$ and it replaces $T \cap X_B$, which is also an $\{A, R\}$ -separator, we find that $|S| \leq |T| < |S'|$.

Since S' was assumed to be a minimum-size OCT of G' , the smaller set S is not an OCT of G' . Therefore, $G' - S$ contains an odd cycle F' . The argument used before to convert an odd cycle in G to one in G' can also be used in the reverse direction to construct an odd cycle F in $G - S$ from F' . The existence of this cycle contradicts the assumption that S is an OCT of G , which concludes the proof by showing that S' is a minimum-size OCT of G . ◀

6 Finding and Removing Tight OCCs

Now we find tight OCCs by the same color coding technique used in previous work [4]. Consider a coloring $\chi: V(G) \cup E(G) \rightarrow \{\dot{B}, \dot{C}, \dot{R}\}$ of the vertices and edges of a graph G . For every color $c \in \{\dot{B}, \dot{C}, \dot{R}\}$, let $\chi_V^{-1}(c) = \chi^{-1}(c) \cap V(G)$. For any integer $z \geq 0$, a z -tight OCC (A_B, A_C, A_R) is z -properly colored by a coloring χ if all the following hold: (i) $A_C \subseteq \chi_V^{-1}(\dot{C})$, (ii) $A_B \subseteq \chi_V^{-1}(\dot{B})$, and (iii) for each component H of $G' = G[A_B \cup A_C] - \chi^{-1}(\dot{R})$ we have $\text{oct}(H) = |A_C \cap V(H)|$ and $|A_C \cap V(H)| \leq z$. Note that $\chi^{-1}(\dot{R})$ may include both vertices and edges, so that the process of obtaining G' involves removing both the vertices and edges colored \dot{R} . By a straight-forward adaptation of the color coding approach from previous work [4, Lemma 6.2], we can reconstruct a tight OCC from a proper coloring.

► **Lemma 18 (★).** *There is an $n^{\mathcal{O}(z)}$ time algorithm taking as input an integer $z \geq 0$, a graph G , and a coloring $\chi: V(G) \cup E(G) \rightarrow \{\dot{B}, \dot{C}, \dot{R}\}$ that either determines that χ does not z -properly color any z -tight OCC, or outputs a z -tight OCC (A_B, A_C, A_R) in G such that for each OCC $(\hat{A}_B, \hat{A}_C, \hat{A}_R)$ that is z -properly colored by χ , we have $\hat{A}_B \subseteq A_B$ and $\hat{A}_C \subseteq A_C$.*

Combining all ingredients in the previous sections leads to a proof of the main theorem.

► **Theorem 1.** *There is a deterministic algorithm that, given a graph G and integers $k \geq z \geq 0$, runs in $2^{\mathcal{O}(k^{33}z^2)} \cdot n^{\mathcal{O}(z)}$ time and either outputs at least k vertices that belong to an optimal solution for ODD CYCLE TRANSVERSAL, or concludes that G does not contain a z -tight OCC of width k .*

Proof sketch. Given an input graph G , we repeatedly invoke Lemma 14 to find a reducible OCC and use the Reduction step to shrink it. When we stabilize on a graph G' , Lemma 16 guarantees that G' contains a z -tight OCC of width $|A'_C| = k$ if G had one. By Lemma 9, there is such a z -tight OCC (A'_B, A'_C, A'_R) in G' for which $G'[A'_B]$ has at most z^2k components. As each such component gives rise to a single-component OCC, none of them are large enough to be reducible. Hence $|A'_C \cup A'_B| \in (zk)^{\mathcal{O}(1)}$. Hence we can deterministically construct a family of $2^{(kz)^{\mathcal{O}(1)}} n^{\mathcal{O}(1)}$ colorings that includes one that properly colors (A'_B, A'_C, A'_R) . Invoking Lemma 18 with such a coloring identifies a z -tight OCC in G' whose head A_C^* contains A'_C and therefore has size at least k . Then A_C^* is contained in an optimal OCT in G' , so that Lemma 17 ensures A_C^* belongs to an optimal OCT in G . We output A_C^* . ◀

7 Conclusion

Inspired by crown decompositions for VERTEX COVER and antler decompositions for FEEDBACK VERTEX SET, we introduced the notion of (tight) odd cycle cuts to capture local regions of a graph in which a simple certificate exists for the membership of certain vertices in an optimal solution to ODD CYCLE TRANSVERSAL. In addition, we developed a fixed-parameter tractable algorithm to find a non-empty subset of vertices that belong to an optimal odd cycle transversal in input graphs admitting a tight odd cycle cut; the parameter k we employed is the *width* of the tight OCC. Finding tight odd cycle cuts and removing the vertices certified to be in an optimal solution leads to search-space reduction for the natural parameterization of ODD CYCLE TRANSVERSAL. To obtain our results, one of the main technical ideas was to replace the use of minimum two-way separators that arise naturally when solving ODD CYCLE TRANSVERSAL, by minimum three-way separators that simultaneously handle breaking the odd cycles in a subgraph *and* separating the resulting local bipartite subgraph from the remainder of the graph.

Theoretical challenges. There are several interesting directions for follow-up work. We first discuss the theoretical challenges. The algorithm we presented runs in time $2^{k^{\mathcal{O}(1)}} n^{\mathcal{O}(z)}$, where z is the order of the tight odd cycle cut in the output guarantee of Theorem 1. The polynomial term in the exponent has a large degree, which is related to the size of the cut covering sets used to shrink the bipartite part of an odd cycle cut in terms of its width. While we expect that some improvements can be made by a more refined analysis, it would be more interesting to see whether an algorithmic approach that avoids color coding can lead to significantly faster algorithms.

An odd cycle cut (X_B, X_C, X_R) of width $|X_C| = k$ in a graph G gives rise to a k -secluded bipartite subgraph $G[X_B]$; recall that a subgraph is called k -secluded if its open neighborhood has size k . For enumerating inclusion-maximal *connected* k -secluded subgraphs that satisfy a property Π , a bounded-depth branching strategy was recently proposed [10] that generalizes the enumeration of important separators. Can such branching techniques be used to improve the running time for the search-space reduction problem considered in this paper to $2^{\mathcal{O}(k)} n^{\mathcal{O}(z)}$?

The dependence on the complexity z of the certificate is another topic for further investigation. The search-space reduction algorithm for FEEDBACK VERTEX SET by Donkers and Jansen [4] that inspired this work, also incurs a factor $n^{\mathcal{O}(z)}$ in its running time. For FEEDBACK VERTEX SET, it is conjectured but not proven that such a dependence on z is unavoidable. The situation is the same for ODD CYCLE TRANSVERSAL. Is there a way to rule out the existence of an algorithm for the task of Theorem 1 that runs in time $f(k, z) \cdot n^{\mathcal{O}(1)}$?

A last theoretical challenge concerns the definition of the substructures that are used to certify membership in an optimal odd cycle transversal. Our definition of an odd cycle cut (X_B, X_C, X_R) prohibits the existence of any edges between X_B and X_R . Together with the requirement that $G[X_B]$ is bipartite, this ensures that all odd cycles intersecting X_B are intersected by X_C . In principle, one could also obtain the latter conclusion from a slightly less restricted graph decomposition. Since any odd cycle enters a bipartite subgraph on one edge and leaves via another, knowing that each connected component H of $G[X_B]$ is connected to X_R by at most one edge is sufficient to guarantee that all odd cycles visiting X_B are intersected by X_C . The prior work on antler structures for FEEDBACK VERTEX SET allows the existence of one pendant edge per component, and manages to detect such antler structures efficiently. It would be interesting to see whether our approach can be generalized for *relaxed* odd cycle cuts in which each component of $G[X_B]$ has at most one edge to X_R . To adapt to this setting, one would have to refine the type of three-way separation problem that is used in the graph reduction step.

For ODD CYCLE TRANSVERSAL, one could relax the definition of the graph decomposition even further: to ensure that odd cycles visiting X_B are intersected by X_C , it would suffice for each connected component H of $G[X_B]$ to have at most one neighbor v_H in X_R , as long as all vertices of H adjacent to v_H belong to the *same* side of a bipartition of H .

Practical challenges. Since the investigation of search-space reduction is inspired by practical considerations, we should not neglect to discuss practical aspects of this research direction. While we do not expect the algorithm as presented here to be practical, it serves as a proof of concept that rigorous guarantees on efficient search-space reduction can be formulated. Our work also helps to identify the types of substructures that can be used to reason locally about membership in an optimal solution. Apart from finding faster algorithms in theory and experimenting with their results, one could also target the development of specialized algorithms for concrete values of k and z .

For $k = 1$, a tight odd cycle cut of width 1 effectively consists of a cutvertex c of the graph whose removal splits off a bipartite connected component B but for which the subgraph induced by $B \cup \{c\}$ contains an odd cycle. Preliminary investigations suggest that in this case, an algorithm that computes the block-cut tree, analyzes which blocks form non-bipartite subgraphs, and which cut vertices break all the odd cycles in their blocks, can be engineered to run in time $\mathcal{O}(|V(G)| + |E(G)|)$ to find a vertex v belonging to an optimal odd cycle transversal when given a graph that has a tight odd cycle cut of width $k = 1$. Do linear-time algorithms exist for $k > 1$? These would form valuable reduction steps in algorithms solving ODD CYCLE TRANSVERSAL exactly, such as the one developed by Wernicke [21].

The $k = 1$ case of the *relaxed* odd cycle cuts described above are in fact used as one of the reduction rules in Wernicke’s algorithm [21, Rule 7]. His reduction applies whenever there is a triangle $\{u, v, w\}$ in which w has degree two and v has degree at most three. Under these circumstances, there is an optimal solution that contains u while avoiding v and w : since the removal of u decreases the degree of w to one, while w is one of the at most two remaining neighbors of v , the removal of u breaks all odd cycles intersecting $\{u, v, w\}$. This corresponds to the fact that the triple $(X_B = \{v, w\}, X_C = \{u\}, X_R = V(G) \setminus \{u, v, w\})$ forms a tight *relaxed* odd cycle cut. We interpret the fact that the $k = 1$ case was developed naturally in an existing algorithm as encouraging evidence that refined research into search-space reduction steps can eventually lead to impact in practice.

References



- 1 Faisal N. Abu-Khzam, Michael R. Fellows, Michael A. Langston, and W. Henry Suters. Crown structures for vertex cover kernelization. *Theory Comput. Syst.*, 41(3):411–430, 2007. doi:10.1007/s00224-007-1328-0.
- 2 Benjamin Merlin Bumpus, Bart M. P. Jansen, and Jari J. H. de Kroon. Search-space reduction via essential vertices. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *Proceedings of the 30th Annual European Symposium on Algorithms, ESA 2022*, volume 244 of *LIPICs*, pages 30:1–30:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ESA.2022.30.
- 3 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
- 4 Huib Donkers and Bart M.P. Jansen. Preprocessing to reduce the search space: Antler structures for feedback vertex set. *Journal of Computer and System Sciences*, 144, 2024. doi:10.1016/j.jcss.2024.103532.
- 5 Michael R. Fellows. Blow-ups, win/win’s, and crown rules: Some new directions in FPT. In Hans L. Bodlaender, editor, *Proceedings of the 29th International Workshop on Graph-theoretic Concepts in Computer Science, WG 2003*, volume 2880 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2003. doi:10.1007/978-3-540-39890-5_1.
- 6 Timothy D. Goodrich, Eric Horton, and Blair D. Sullivan. An updated experimental evaluation of graph bipartization methods. *ACM J. Exp. Algorithmics*, 26:12:1–12:24, 2021. doi:10.1145/3467968.
- 7 Timothy D. Goodrich, Blair D. Sullivan, and Travis S. Humble. Optimizing adiabatic quantum program compilation using a graph-theoretic framework. *Quantum Information Processing*, 17(5), April 2018. doi:10.1007/s11128-018-1863-4.
- 8 Falk Hüffner. Algorithm engineering for optimal graph bipartization. *J. Graph Algorithms Appl.*, 13(2):77–98, 2009. URL: <http://jgaa.info/accepted/2009/Hueffner2009.13.2.pdf>, doi:10.7155/JGAA.00177.
- 9 Bart M. P. Jansen and Jari J. H. de Kroon. FPT algorithms to compute the elimination distance to bipartite graphs and more. In Lukasz Kowalik, Michal Pilipczuk, and Pawel Rzazewski, editors, *Graph-Theoretic Concepts in Computer Science - 47th International Workshop, WG 2021, Warsaw, Poland, June 23-25, 2021, Revised Selected Papers*, volume 12911 of *Lecture Notes in Computer Science*, pages 80–93. Springer, 2021. doi:10.1007/978-3-030-86838-3_6.
- 10 Bart M. P. Jansen, Jari J. H. de Kroon, and Michal Włodarczyk. Single-exponential FPT algorithms for enumerating secluded $\{$ -free subgraphs and deleting to scattered graph classes. In Satoru Iwata and Naonori Kakimura, editors, *Proceedings of the 34th International Symposium on Algorithms and Computation, ISAAC 2023*, volume 283 of *LIPICs*, pages 42:1–42:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.ISAAC.2023.42.
- 11 Bart M. P. Jansen, Jari J. H. de Kroon, and Michał Włodarczyk. Vertex Deletion Parameterized by Elimination Distance and Even Less, 2022. arXiv:2103.09715 [cs]. doi:10.48550/arXiv.2103.09715.
- 12 Bart M. P. Jansen, Yosuke Mizutani, Blair D. Sullivan, and Ruben F. A. Verhaegh. Preprocessing to reduce the search space for odd cycle transversal, 2024. arXiv:2409.00245, doi:10.48550/arXiv.2409.00245.
- 13 Bart M. P. Jansen and Ruben F. A. Verhaegh. Search-space reduction via essential vertices revisited: Vertex multicut and cograph deletion. In Hans L. Bodlaender, editor, *Proceedings of the 19th Scandinavian Symposium on Algorithm Theory, SWAT 2024*, LIPICs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. In press. arXiv:2404.09769, doi:10.4230/LIPICs.SWAT.2024.28.
- 14 Stefan Kratsch and Magnus Wahlström. Compression via matroids: A randomized polynomial kernel for odd cycle transversal. *ACM Trans. Algorithms*, 10(4):20:1–20:15, 2014. doi:10.1145/2635810.

- 15 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. *J. ACM*, 67(3):16:1–16:50, 2020. doi:10.1145/3390887.
- 16 Daniel Lokshantov, N. S. Narayanaswamy, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. Faster parameterized algorithms using linear programming. *ACM Trans. Algorithms*, 11(2):15:1–15:31, 2014. doi:10.1145/2566616.
- 17 Moni Naor, Leonard J. Schulman, and Aravind Srinivasan. Splitters and near-optimal derandomization. In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995*, pages 182–191. IEEE Computer Society, 1995. doi:10.1109/SFCS.1995.492475.
- 18 Marcin Pilipczuk and Michal Ziobro. Experimental evaluation of parameterized algorithms for graph separation problems: Half-integral relaxations and matroid-based kernelization. *CoRR*, abs/1811.07779, 2018. arXiv:1811.07779.
- 19 Bruce A. Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Oper. Res. Lett.*, 32(4):299–301, 2004. doi:10.1016/j.orl.2003.10.009.
- 20 Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.
- 21 Sebastian Wernicke. *On the algorithmic tractability of single nucleotide polymorphism (SNP) analysis and related problems*. diplom.de, 2014.

Modularity Clustering Parameterized by Max Leaf Number

Jaroslav Garvardt  

Institute of Computer Science, Friedrich Schiller University Jena, Germany

Christian Komusiewicz  

Institute of Computer Science, Friedrich Schiller University Jena, Germany

Abstract

The modularity score is one of the most important measures for assessing the quality of clusterings of undirected graphs. In the notoriously difficult MODULARITY problem, one is given an undirected graph G and the task is to find a clustering with maximum modularity. We show that MODULARITY is fixed-parameter tractable with respect to the max leaf number of G . This improves on a previous result by Meeks and Skerman [Algorithmica '20] who showed an XP-algorithm for this parameter. In addition, we strengthen previous hardness results for MODULARITY by showing W[1]-hardness for the parameter vertex deletion distance to disjoint union of stars.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Graph clustering, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.16

Funding Jaroslav Garvardt: Supported by the Carl Zeiss Foundation, Germany, within the project “Interactive Inference”.

1 Introduction

One of the most central topics in network science is the detection of community structure [18]. This can be achieved via graph clustering. In its most simple form, this is the task of searching for a partition of the vertex set into clusters and the underlying assumption is that edges are more likely to be present inside clusters than between them. The corresponding optimization goal is to maximize the edge coverage of the partition, that is, the number of intracluster edges. Of course, the trivial partition into one single cluster trivially maximizes the number of intracluster edges. To counter this, several approaches have been proposed, for example one may demand that clusters form highly connected subgraphs [11, 12] or one may penalize missing edges inside clusters [19].

The arguably most popular way of achieving clusterings with high edge coverage while avoiding the trivial clustering is to maximize modularity [17]. In the modularity measure, the contribution of a cluster consists of two parts: the first part corresponds to the edge coverage and the second part is a degree tax which penalizes clusters that have many high-degree vertices. The idea behind the degree tax is that such clusters are expected to contain many edges simply because there are many edges that are incident with the cluster vertices. Hence, for a positive contribution to the modularity score, the number of present edges should exceed the number of expected ones. More formally, a clustering of a graph $G = (V, E)$ is a partition of V and for a vertex set $C \subseteq V$, $E(C)$ denotes the set of edges with both endpoints in C . Now the definition of modularity reads as follows.

► **Definition 1.1.** *The modularity of a clustering \mathcal{C} of a graph G with m edges is given by*

$$q(\mathcal{C}) = \sum_{C \in \mathcal{C}} \frac{|E(C)|}{m} - \frac{(\sum_{v \in C} \deg(v))^2}{4m^2}.$$



© Jaroslav Garvardt and Christian Komusiewicz;
licensed under Creative Commons License CC-BY 4.0

19th International Symposium on Parameterized and Exact Computation (IPEC 2024).

Editors: Édouard Bonnet and Paweł Rzażewski; Article No. 16; pp. 16:1–16:14

Leibniz International Proceedings in Informatics



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

16:2 Modularity Clustering Parameterized by Max Leaf Number

The problem of computing an optimal clustering under this measure is now defined as follows.

MODULARITY

Input: A graph $G = (V, E)$.

Task: Find a clustering \mathcal{C} with maximum modularity $q(\mathcal{C})$.

MODULARITY is NP-hard [4] and therefore heuristics, for example greedy algorithms [5] or local search [3], are prevalent in practice. The modularity measure has some counterintuitive behavior [4]. Consequently, some research focuses on better understanding the properties of optimal clusterings [4] or on providing bounds for the modularity values of certain graph classes [1, 14, 15, 8, 20]; for an overview, refer to the work of Skerman [20].

The importance of the modularity measure has motivated further research into the complexity of MODULARITY. For example, computing the best clustering with exactly two clusters is also NP-hard [4] even when the input graphs are restricted to be d -regular for any $d \geq 9$ [7]. Meeks and Skerman [16] initiated the analysis of MODULARITY within the framework of parameterized complexity obtaining the following results: They showed that MODULARITY can be solved in polynomial time on graphs with constant treewidth and that MODULARITY is fixed-parameter tractable with respect to the vertex cover number of the input graph G . Moreover, it was shown that MODULARITY is W[1]-hard with respect to the parameter pathwidth of G plus feedback vertex set number of G , so presumably there exists no FPT-algorithm for this parameter. For the parameter max leaf number of G , denoted $\lambda(G)$, an XP-algorithm was shown. That is, the algorithm has a polynomial running time for constant values of $\lambda(G)$, but the degree of the polynomial depends on $\lambda(G)$. The precise parameterized complexity of MODULARITY with respect to the max leaf number of G was left open.

In this work, we continue this line of research. We show that the XP-algorithm for the max leaf number $\lambda(G)$ can be improved to an FPT-algorithm. While the max leaf number, one of the most classic structural parameters [10], is quite restrictive, our result provides only the second nontrivial FPT-algorithm for this very important problem. Roughly speaking, our algorithm exploits that large graphs with bounded max leaf number contain very long paths consisting of degree-2 vertices and that the clustering for these paths follows a relatively regular pattern. The algorithm consists of three steps. In a first branching, the global structure of the clustering is constrained. In particular, it is determined how the vertices of degree at least 3 are clustered and with which degree-2 paths these clusters share vertices. To prepare the next step, it is shown that the clusters consisting of degree-2 vertices have roughly the same size and, based on this, that the clusters containing high-degree vertices also deviate by at most $22\lambda(G)^2$ from the size of the path clusters. This allows us to find the correct cluster sizes via branching. The remaining problem of computing an optimal clustering under these size constraints is then solved via an ILP formulation.

Our algorithm works also on disconnected graphs G , where we define $\lambda(G)$ to be the sum of the max leaf numbers of the connected components of G . The only proofs where we assume connectivity are those that bound the number of high-degree vertices in terms of $\lambda(G)$ and they are easily seen to also hold for disconnected graphs by summing over the connected components.

On the negative side, we strengthen the previous W[1]-hardness for MODULARITY parameterized by pathwidth plus feedback vertex set number by showing that MODULARITY is W[1]-hard with respect to the vertex deletion distance of G to a disjoint union of stars. This parameter is obviously at least as large as the feedback vertex set number of G . Moreover, the parameter is also lower-bounded by pathwidth + 1: Any vertex deletion set S to a disjoint union of stars gives a path decomposition where every bag contains S plus a star center plus one leaf of the star.

In our opinion, this $W[1]$ -hardness for distance to stars puts the FPT-algorithm for the admittedly large parameter max leaf number into context by underlining once more that MODULARITY is resistant to quite large structural parameterizations. Due to space constraints for statements marked with a star (*), the proofs are deferred to the long version of this work.

2 Preliminaries

We consider undirected graphs $G = (V, E)$ and let n denote the number of vertices of G and m the number of edges of G . The *neighborhood* of a vertex $v \in V$ is defined as $N(v) = \{u \in V \mid \{u, v\} \in E\}$ and for a set of vertices $V' \subseteq V$ we define $N(V') = (\bigcup_{v \in V'} N(v)) \setminus V'$. The *degree* of a vertex v is denoted by $\deg(v) = |N(v)|$. We define $V_{=1} = \{v \in V \mid \deg(v) = 1\}$ and analogously $V_{=2} = \{v \in V \mid \deg(v) = 2\}$ and $V_{\geq 3} = \{v \in V \mid \deg(v) \geq 3\}$.

We denote the degree sum of a vertex set C , also called *volume* of C , by $\text{vol}(C) := \sum_{v \in C} \deg(v)$. For two clusterings \mathcal{C} and \mathcal{C}' we say that \mathcal{C} is *better* than \mathcal{C}' if $q(\mathcal{C}) > q(\mathcal{C}')$. Since we are often not interested in the actual value of the modularity of a clustering, but only whether it is better than another clustering, we define the function $\tilde{q}(\mathcal{C}) = 4m^2q(\mathcal{C}) = 4m \sum_{C \in \mathcal{C}} |E(C)| - \sum_{C \in \mathcal{C}} \text{vol}(C)^2$. Clearly, for two clusterings \mathcal{C} and \mathcal{C}' of the same graph we have $q(\mathcal{C}) \geq q(\mathcal{C}')$ if and only if $\tilde{q}(\mathcal{C}) \geq \tilde{q}(\mathcal{C}')$.

A *2-path* is a path (v_1, v_2, \dots, v_k) with $\deg(v_i) \leq 2$ for all $i \in [k]$. A 2-path is *maximal* if it is not contained in a longer 2-path. For a maximal 2-path $P = (v_1, v_2, \dots, v_k)$ we refer to v_1 and v_k as the endpoints of P and define $V(P) = \{v_1, v_2, \dots, v_k\}$. A 2-path is *pendent* if $\deg(v_1) = 1$ or $\deg(v_k) = 1$. A *branch* of a graph G is a maximal path or cycle in which every *internal* vertex of the path has degree 2 in G . We denote with \mathcal{B}_G the set of all branches in G and with $\beta(G) = |\mathcal{B}_G|$ the number of all branches in G . Note that $\beta(G)$ can be computed in $\mathcal{O}(n + m)$ time. Let G be a connected graph. The *maximum leaf number* $\lambda(G)$ of G (or just *max leaf number*) is the maximum number of leaves in any spanning tree of G . When the graph G is clear from the context we just write λ for the max leaf number.

► **Lemma 2.1** ([4]). *There is always a clustering with maximum modularity, in which each cluster induces a connected subgraph.*

► **Lemma 2.2** ([4]). *A clustering with maximum modularity has no cluster that consists of a single vertex with degree 1.*

In our correctness proofs, we are often concerned with the effect of removing one vertex from some cluster C_i and adding some vertex to another cluster C_j . In particular, we are interested in the change of the total degree tax for these two clusters. The following lemma describes a situation where the degree tax decreases.

► **Lemma 2.3.** *Let C_i and C_j be two clusters and let u and v be two vertices of the same degree such that $u \in C_i$ and $\deg(u) > 0$. If $\text{vol}(C_i \setminus \{u\}) > \text{vol}(C_j)$, then $\text{vol}(C_i)^2 + \text{vol}(C_j)^2 > \text{vol}(C_i \setminus \{u\})^2 + \text{vol}(C_j \cup \{v\})^2$.*

Proof. The claim holds trivially if $v \in C_j$, thus assume $v \notin C_j$. Since $u \in C_i$, we have $\text{vol}(C_i) = \text{vol}(C_i \setminus \{u\}) + \deg(u)$. Therefore,

$$\text{vol}(C_i)^2 = (\text{vol}(C_i \setminus \{u\}) + \deg(u))^2 = \text{vol}(C_i \setminus \{u\})^2 + 2 \cdot \text{vol}(C_i \setminus \{u\}) \cdot \deg(u) + \deg(u)^2.$$

Since $\deg(u) = \deg(v)$, we similarly have

$$\text{vol}(C_j \cup \{v\})^2 = (\text{vol}(C_j) + \deg(u))^2 = \text{vol}(C_j)^2 + 2 \cdot \text{vol}(C_j) \cdot \deg(u) + \deg(u)^2.$$

16:4 Modularity Clustering Parameterized by Max Leaf Number

Thus,

$$\begin{aligned} & \text{vol}(C_i)^2 + \text{vol}(C_j)^2 - (\text{vol}(C_i \setminus \{u\})^2 + \text{vol}(C_j \cup \{v\})^2) \\ &= 2 \cdot \text{vol}(C_i \setminus \{u\}) \cdot \text{deg}(u) + \text{deg}(u)^2 - (2 \cdot \text{vol}(C_j) \cdot \text{deg}(u) + \text{deg}(u)^2) \\ &= 2 \cdot \text{deg}(u) \cdot (\text{vol}(C_i \setminus \{u\}) - \text{vol}(C_j)) > 0. \end{aligned} \quad \blacktriangleleft$$

We now show that the number of vertices $v \in V$ with $\text{deg}(v) \geq 3$ is bounded by a function of the max leaf number. More precisely, we give a bound on the sum of the degrees of these vertices. This bound is obtained via bounding the number of branches in G . Eppstein [9] already showed that this number is $\mathcal{O}(\lambda(G)^2)$. We give a precise bound on the hidden constant since we will use it in our algorithm.

► **Lemma 2.4 (*)**. *Let $G = (V, E)$ be a connected graph. Then we have $\beta(G) \leq 21 \lambda(G)^2$.*

The next statement directly follows from Lemma 2.4, since each edge incident with a vertex $v \in V_{\geq 3}$ corresponds to a branch and each branch contains at most two vertices in $V_{\geq 3}$.

► **Corollary 2.5**. *Let $G = (V, E)$ be a connected graph. Then we have $\sum_{v \in V_{\geq 3}} \text{deg}(v) \leq 42 \lambda(G)^2$.*

3 Preclusterings and Cluster Sizes

3.1 Preclustering Branching

The general approach of the algorithm is to consider in a branching step all the possibilities of how some optimal clustering might interact with the vertices of degree at least 3. The structure containing this information is a partial clustering defined as follows.

► **Definition 3.1**. *A preclustering is a set $\mathcal{P} := \{C_1, \dots, C_s\}$ of nonempty disjoint subsets of V .*

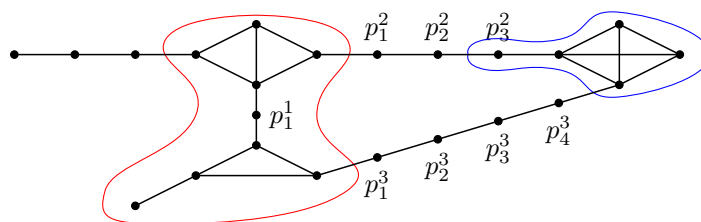
► **Definition 3.2**. *A clustering $\mathcal{C} = \{C'_1, \dots, C'_t\}$ extends a preclustering $\mathcal{P} = \{C_1, \dots, C_s\}$ if for each $C_i \in \mathcal{P}$ there is exactly one cluster $C'_j \in \mathcal{C}$ such that $C_i \subseteq C'_j$ and each cluster $C'_j \in \mathcal{C}$ has nonempty intersection with at most one cluster of \mathcal{P} .*

To find an optimal solution efficiently from a preclustering, we will need to fix not only which vertices of degree 3 are contained in which clusters but also how these clusters interact with the potentially very long 2-paths connecting them. The necessary information is provided by what we call full preclusterings, defined as follows (see Figure 1 for an example).

► **Definition 3.3**. *A preclustering $\mathcal{P} = \{C_1, \dots, C_s\}$ is a full preclustering if every vertex of $V_{\geq 3}$ is contained in some cluster of \mathcal{P} and for every maximal 2-path $P = (v_1, v_2, \dots, v_k)$ of G either*

- *all vertices of P are contained in some common cluster C_i ,*
- *no vertex of P is contained in any cluster C_i , or*
- *$k \geq 2$, for each cluster C_i we have $C_i \cap V(P) \subseteq \{v_1, v_k\}$ and if $v_1 \in C_i$ then also $u_1 \in C_i$, where u_1 is the unique neighbor of v_1 in $V_{\geq 3}$, and if $v_k \in C_i$ then also $u_k \in C_i$, where u_k is the unique neighbor of v_k in $V_{\geq 3}$.*

The idea of full preclusterings is as follows. For the fully contained maximal 2-paths P , the cluster is already fixed. For the endpoints of the other maximal 2-paths, we know that they are either 1) in different clusters than their high-degree neighbors which helps us to separate the instance in smaller pieces, or 2) in the same cluster as their high-degree neighbors which allows us to use the clusters with these high-degree vertices in some exchange arguments because they also contain some degree-2 vertices.



■ **Figure 1** Example of a full preclustering $\mathcal{P} = \{C_1, C_2\}$. The cluster C_1 is encircled in red, the cluster C_2 is encircled in blue. Cluster C_1 contains the complete maximal 2-path (p_1^1) , cluster C_2 contains one endpoint of the maximal 2-path (p_1^2, p_2^2, p_3^2) and no vertex of the maximal 2-path $(p_1^3, p_2^3, p_3^3, p_4^3)$ is contained in any cluster in \mathcal{P} .

- **Definition 3.4.** A clustering \mathcal{C} legally extends a full preclustering \mathcal{P} if
- \mathcal{C} extends \mathcal{P} ,
 - a cluster $C \in \mathcal{C}$ contains $\{u, v\}$ where u is an endpoint of a maximal 2-path P and v is a neighbor of u in $V_{\geq 3}$ only if some cluster of \mathcal{P} does.

Note that for a given clustering \mathcal{C} , there is exactly one full preclustering \mathcal{P} such that \mathcal{C} legally extends \mathcal{P} . We say that \mathcal{P} is the preclustering that *corresponds* to \mathcal{C} .

Let us first show that we may indeed consider all full preclusterings within FPT time.

- **Lemma 3.5.** Any graph G has $\lambda(G)^{\mathcal{O}(\lambda(G)^2)}$ full preclusterings.

Proof. A full preclustering \mathcal{P} can be identified by

1. the partition of $V_{\geq 3}$ that it induces,
2. for each 2-path, the information whether that 2-path is fully contained in some cluster of \mathcal{P} , disjoint from all clusters of \mathcal{P} , or whether its endpoints are contained in some cluster of \mathcal{P} .

In the latter case, the cluster which contains an endpoint is uniquely determined to be the cluster containing the neighbor of the endpoint in $V_{\geq 3}$. By Corollary 2.5, the number of vertices in $V_{\geq 3}$ is $\mathcal{O}(\lambda(G)^2)$ and thus the number of partitions of $V_{\geq 3}$ is $\lambda(G)^{\mathcal{O}(\lambda(G)^2)}$. By Lemma 2.4, the number of branches and thus the number of 2-paths is $\mathcal{O}(\lambda(G)^2)$. For each 2-path, we need to distinguish altogether five cases, hence there are $2^{\mathcal{O}(\lambda(G)^2)}$ possibilities for the 2-path information. The total number of full preclusterings is thus $\lambda(G)^{\mathcal{O}(\lambda(G)^2)} \cdot 2^{\mathcal{O}(\lambda(G)^2)} = \lambda(G)^{\mathcal{O}(\lambda(G)^2)}$. ◀

A full preclustering \mathcal{P} constrains some edges of the graph to not be contained in any cluster of a clustering that legally extends \mathcal{P} . This set of edges is defined as follows.

- **Definition 3.6.** Let $\mathcal{P} = \{C_1, \dots, C_s\}$ be a full preclustering of G . The separation induced by \mathcal{P} is the edge set

$$S(\mathcal{P}) := \bigcup_{i \in [s]} \{\{u, v\} \in E \mid u \in C_i \cap V_{\geq 3} \text{ and } v \notin C_i\}.$$

As the name suggests a separation fully separates some parts of the instance. These are exactly the connected components of $G - S(\mathcal{P})$, they are called the *separated components* of \mathcal{P} . By Lemma 2.1 it is sufficient to consider clusterings such that every cluster induces a connected subgraph. For any such clustering \mathcal{C} that legally extends a full preclustering \mathcal{P} , we have that every cluster C is completely contained in some separated component of \mathcal{P} . We thus compute an optimal clustering of each separated component of \mathcal{P} individually.

16:6 Modularity Clustering Parameterized by Max Leaf Number

We will distinguish those clusters that contain at least one vertex from $V_{\geq 3}$, these are called *base clusters*, from those clusters that are contained in the 2-paths, these are called *path clusters*. The two main parts that are not yet determined by a full preclustering are how far each base cluster extends into the neighboring 2-paths and how large the clusters which are fully contained in 2-paths are. The next step is now to show that the 2-path clusters inside a separated component have roughly the same size. Note that this is not true for all full preclusterings but rather that there is some preclustering which has a globally optimal legal extension for which this is the case.

- **Lemma 3.7.** *There exists an optimal clustering \mathcal{C} such that*
 - *\mathcal{C} legally extends some full preclustering \mathcal{P} , and*
 - *in every separated component S of \mathcal{P} , there is some number p such that the path clusters in S have size p or $p + 1$.*

The approach to show Lemma 3.7 is, roughly speaking, to show that a big size difference between path clusters leads to suboptimality because we can exchange some degree-2 vertices to balance the cluster sizes. This exchange may need to involve base clusters which contain some vertices of 2-paths. To distinguish whether a base cluster contains some vertices of a 2-path or not, we say a base cluster C *extends* into a 2-path P if $|C \cap P| \geq 1$. Note that per definition a base cluster C extends into a 2-path P if and only if in the corresponding preclustering there is a cluster $C_P \subseteq C$ such that $|C_P \cap P| \geq 1$.

- **Definition 3.8.** *Two clusters $C \in \mathcal{C}$ and $C' \in \mathcal{C}$ are neighboring clusters or neighbors if $\{u, v\} \in E$ for some $u \in C, v \in C'$.*

For a clustering \mathcal{C} with neighboring path clusters $C_1 = \{u_i, \dots, u_j\}$ and $C_2 = \{u_{j+1}, \dots, u_{j+\ell}\}$ on a 2-path $P = (u_1, \dots, u_t)$, we define the clustering \mathcal{C}' obtained by the *swap* of C_1 and C_2 as the clustering that is the same as \mathcal{C} except for clusters C_1 and C_2 which are replaced by the clusters $C'_1 = \{u_i, \dots, u_{i+\ell-1}\}$ and $C'_2 = \{u_{i+\ell}, \dots, u_{j+\ell}\}$. In other words, the swap exchanges the lengths of two neighboring path clusters. Clearly, the clustering resulting from applying a swap to \mathcal{C} has the same modularity as \mathcal{C} .

3.2 Difference in Cluster Sizes is bounded in λ

We now prove a series of lemmas which are needed for the proof of Lemma 3.7. We distinguish those path clusters that contain a degree-1 vertex which we call *pendent* path clusters and those that do not contain a degree-1 vertex which are called *nonpendent* path clusters.

Note that a separated component consisting of a 2-path with two vertices of degree 1 is an isolated path, a graph with constant treewidth, for which the optimal clustering can be computed directly in polynomial time [16]. We therefore assume in the following that there is at most one pendent path cluster per 2-path.

The first lemma shows that in optimal solutions, pendent clusters are at least as large as neighboring nonpendent clusters.

- **Lemma 3.9 (*)**. *Let $G = (V, E)$ be a graph and \mathcal{C} a clustering of G . Let P be a pendent 2-path and $C_1 \in \mathcal{C}$ and $C_2 \in \mathcal{C}$ be path clusters in P such that C_2 is pendent, C_1 is nonpendent, and $|C_1| > |C_2|$. Then, \mathcal{C} is not optimal.*

The next lemma shows that path clusters from the same 2-path in G can only differ in size by at most one vertex.

- **Lemma 3.10.** *Let $G = (V, E)$ be a graph and \mathcal{C} a clustering of G . Let P be a 2-path and $C_1 \in \mathcal{C}$ and $C_2 \in \mathcal{C}$ be path clusters in P with $|C_1| > |C_2| + 1$. Then, \mathcal{C} is not optimal.*

Proof. Recall that we can assume that not both of C_1 and C_2 are pendent, since otherwise P is an isolated path for which the optimal clustering can be computed directly in polynomial time. Since P is a 2-path, we can swap neighboring clusters of P without changing the modularity, so we can assume that C_1 and C_2 are neighboring clusters.

Let $v_1 \in C_1 \cap N(C_2)$ be the neighbor of C_2 in C_1 . Consider the clustering \mathcal{C}' where C_1 and C_2 are replaced by clusters $C'_1 = C_1 \setminus \{v_1\}$ and $C'_2 = C_2 \cup \{v_1\}$, respectively, and all other clusters are unchanged.

Note that we only have to consider the contribution of C_1 and C_2 to $\tilde{q}(\mathcal{C})$ and the contribution of C'_1 and C'_2 to $\tilde{q}(\mathcal{C}')$ since the other clusters are identical for both clusterings. Furthermore, since v_1 is part of a 2-path, we have $|E(C'_1)| = |E(C_1)| - 1$ and $|E(C'_2)| = |E(C_2)| + 1$, so the total number of intracluster edges remains the same.

Moreover, by Lemma 3.9, we may assume that C_2 is nonpendent. Now, if C_1 is nonpendent, then $\text{vol}(C_1 \setminus \{v_1\}) = 2|C_1| - 2 > 2|C_2| = \text{vol}(C_2)$ since $|C_1| > |C_2| + 1$. Thus Lemma 2.3 implies $\text{vol}(C_1)^2 + \text{vol}(C_2)^2 > \text{vol}(C_1 \setminus \{v_1\})^2 + \text{vol}(C_2 \cup \{v_1\})^2$. Finally, if C_1 is pendent, then again $\text{vol}(C_1 \setminus \{v_1\}) = 2|C_1| - 3 \geq 2|C_2| + 1 > 2|C_2| = \text{vol}(C_2)$ since $|C_1| > |C_2| + 1$ and thus $|C_1| \geq |C_2| + 2$. Hence, \mathcal{C}' is a better clustering. \blacktriangleleft

The next lemma shows that two path clusters with the same base cluster as a neighbor can only differ in size by at most one vertex.

► **Lemma 3.11 (*).** *Let $G = (V, E)$ be a graph and \mathcal{C} a clustering of G . Let $C \in \mathcal{C}$ be a base cluster that extends into a 2-path P_1 and a 2-path P_2 . Let $C_{P_1} \in \mathcal{C}$ be a path cluster in P_1 and $C_{P_2} \in \mathcal{C}$ be a path cluster in P_2 with $|C_{P_1}| > |C_{P_2}| + 1$. Then, \mathcal{C} is not optimal.*

We are now ready to show Lemma 3.7.

► **Lemma 3.7.** *There exists an optimal clustering \mathcal{C} such that*

- \mathcal{C} extends some full preclustering \mathcal{P} , and
- in every separated component of \mathcal{P} , there is some number p such that the path clusters have size p or $p + 1$.

For the proof of Lemma 3.7 we need the following definition.

► **Definition 3.12.** *Let \mathcal{C} be a clustering that extends some full preclustering \mathcal{P} and let S be a separated component of \mathcal{P} . Let C and \tilde{C} be path clusters in S contained in the 2-paths P and \tilde{P} , respectively. Since C and \tilde{C} are part of the same separated component, there is a smallest number $\ell \geq 1$ for which there is a sequence of 2-paths P_1, \dots, P_ℓ and a sequence of extended base clusters $B_1, \dots, B_{\ell-1}$ such that $P = P_1$, $\tilde{P} = P_\ell$ and B_i extends into P_i and P_{i+1} for each $i \in [\ell - 1]$. We then say that C and \tilde{C} have path cluster distance ℓ .*

Proof (of Lemma 3.7). Assume towards a contradiction that for every optimal clustering \mathcal{C} and its corresponding full preclustering \mathcal{P} , there is a separated component of \mathcal{P} that contains path clusters C_1 and C_2 with $|C_1| - |C_2| \geq 2$. We choose \mathcal{C} in such a way that there is a separated component S of \mathcal{P} such that S contains path clusters C and \tilde{C} with

- $p := |C|$ and $\tilde{p} := |\tilde{C}| = p + c$ for some $c > 1$,
- the clusters C and \tilde{C} have path cluster distance r , and
- r is the minimal path cluster distance of any two path clusters C_1, C_2 with $|C_1| - |C_2| \geq 2$ in the same separated component of any optimal clustering.

Let P be the 2-path that contains C and \tilde{P} be the 2-path that contains \tilde{C} . Let P_1, \dots, P_r and B_1, \dots, B_{r-1} be the sequences of 2-paths and extended base clusters for C and \tilde{C} as described in Definition 3.12.

If $r = 1$, then we have $\tilde{p} \in \{p, p+1\}$ by Lemma 3.10 and the optimality of \mathcal{C} , contradicting the assumption $\tilde{p} = p + c$. If $r = 2$, then there is a base cluster B_1 that extends into both P and \tilde{P} , so we have again $\tilde{p} \in \{p, p+1\}$, since otherwise \mathcal{C} would not be optimal according to Lemma 3.11, contradicting the assumption $\tilde{p} = p + c$. Now consider the case $r \geq 3$. First, we show that we can assume that C and B_1 are neighboring clusters: If C is pendent and has a neighboring nonpendent path cluster C' of size $p+1$, then the clustering \mathcal{C} is not optimal by Lemma 3.9. Hence, if C is pendent and not a neighboring cluster of B_1 , then we may choose the nonpendent cluster C' instead of C . Now, if C is nonpendent, since P_1 is a 2-path, we can swap neighboring clusters of P_1 until we reach B_1 without changing the modularity. Altogether, we can assume that C and B_1 are neighboring clusters. Let \hat{C} be the path cluster of P_2 neighboring B_1 . Note that $|\hat{C}| \in \{p-1, p, p+1\}$ due to Lemma 3.11. If $|\hat{C}| \in \{p-1, p\}$, then we have a contradiction to C and \tilde{C} having minimal path cluster distance, since $|\tilde{C}| - |\hat{C}| \geq 2$ and \hat{C} and \tilde{C} have path cluster distance $r-1$. Thus, let $|\hat{C}| = p+1$. Let $v_1 \in B_1 \cap N(C)$ be the neighbor of C in B_1 and let $v_2 \in \hat{C} \cap N(B_1)$ be the neighbor of B_1 in \hat{C} . Consider the clustering \mathcal{C}' where B_1 , C , and \hat{C} are replaced by clusters $B'_1 = (B_1 \setminus \{v_1\}) \cup \{v_2\}$, $C' = C \cup \{v_1\}$, and $\hat{C}' = \hat{C} \setminus \{v_2\}$, respectively. Clearly, we have $\tilde{q}(\mathcal{C}') = \tilde{q}(\mathcal{C})$, so \mathcal{C}' is also an optimal clustering. Note that in \mathcal{C}' the clusters \hat{C}' and \tilde{C} are in the same separated component of the preclustering \mathcal{P}' corresponding to \mathcal{C}' . Moreover, in \mathcal{C}' the clusters \hat{C}' and \tilde{C} have path cluster distance $r-1$ and $|\hat{C}'| = p$. Altogether, we thus have a contradiction to C and \tilde{C} having minimal path cluster distance. \blacktriangleleft

By Lemma 3.7 for each separated component we can distinguish between *small* and *big* path clusters of size p and $p+1$, respectively. The next lemma shows that for an extended base cluster and a neighboring path cluster the size difference is bounded by a function of $\lambda(G)$.

► **Lemma 3.13.** *Let $G = (V, E)$ be a graph and \mathcal{C} a clustering of G . Let $C \in \mathcal{C}$ be a base cluster that extends into a 2-path P and let $C_P \in \mathcal{C}$ be a path cluster in P . If $|C_P| > |C| + 22\lambda(G)^2$ or $|C_P| < |C| - 2$, then \mathcal{C} is not optimal.*

Proof. Without loss of generality, we may assume that C_P is a neighboring cluster of the base cluster C , as otherwise all exchanges between C and C_P can be carried out by moving the path clusters between C and C_P .

First, consider the case where $|C_P| > |C| + 22\lambda(G)^2$. Let $v \in C_P \cap N(C)$ be the neighbor of C in C_P . Consider the clustering \mathcal{C}' where C_P and C get replaced by clusters $C'_P = C_P \setminus \{v\}$ and $C' = C \cup \{v\}$, respectively, and all other clusters are not changed.

Note that we only have to consider the contribution of C_P and C to $\tilde{q}(\mathcal{C})$ and the contribution of C'_P and C' to $\tilde{q}(\mathcal{C}')$ since the other clusters are identical for both clusterings. Furthermore, since v is part of a 2-path, we have $|E(C'_P)| = |E(C_P)| - 1$ and $|E(C')| = |E(C)| + 1$, so the total number of intracluster edges remains the same. We can express the volume of the base cluster C as the sum $\text{vol}(C) = \text{vol}(C \cap V_{=1}) + \text{vol}(C \cap V_{=2}) + \text{vol}(C \cap V_{\geq 3})$. According to Lemma 2.5 we have $\text{vol}(V_{\geq 3}) \leq 42\lambda(G)^2$ and therefore also $\text{vol}(C \cap V_{\geq 3}) \leq 42\lambda(G)^2$. Thus, if C_P is nonpendent, we have

$$\text{vol}(C_P \setminus \{v\}) = 2|C_P| - 2 > 2|C| + 44\lambda(G)^2 - 2 > \text{vol}(C).$$

Similarly, if C_P is pendent, then

$$\text{vol}(C_P \setminus \{v\}) = 2|C_P| - 3 > 2|C| + 44\lambda(G)^2 - 3 > \text{vol}(C).$$

Thus, in both cases Lemma 2.3 implies $\text{vol}(C_P)^2 + \text{vol}(C)^2 > \text{vol}(C_P \setminus \{v\})^2 + \text{vol}(C \cup \{v\})^2$ and \mathcal{C}' is a better clustering.

Second, consider the case $|C_P| < |C| - 2$. Let $v \in C \cap N(C_P)$ be the neighbor of C_P in C . Consider the clustering \mathcal{C}' where C and C_P get replaced by clusters $C' = C \setminus \{v\}$ and $C'_P = C_P \cup \{v\}$, respectively, and all other clusters are not changed.

Again, we only have to consider the contribution of C_P and C to $\tilde{q}(\mathcal{C})$ and the contribution of C'_P and C' to $\tilde{q}(\mathcal{C}')$ since the other clusters are identical for both clusterings. Furthermore, note that since v is part of a 2-path, we have $|E(C'_P)| = |E(C_P)| + 1$ and $|E(C')| = |E(C)| - 1$, so the total number of intracluster edges remains the same. Moreover, since C is connected we have $\text{vol}(C) > 2|C| - 2$. Hence, $\text{vol}(C \setminus \{v\}) > 2|C| - 4 = 2(|C| - 2) > 2|C_P| \geq \text{vol}(C_P)$. Thus, Lemma 2.3 implies $\text{vol}(C)^2 + \text{vol}(C_P)^2 > \text{vol}(C \setminus \{v\})^2 + \text{vol}(C_P \cup \{v\})^2$ and therefore the clustering \mathcal{C} is not optimal. ◀

► **Lemma 3.14.** *Let $G = (V, E)$ be a graph and \mathcal{C} a clustering of G . Let $C \in \mathcal{C}$ and $\widehat{C} \in \mathcal{C}$ be base clusters of the same separated component such that $|C| + 22\lambda(G)^2 < |\widehat{C}|$. Then, \mathcal{C} is not optimal.*

Proof. Let $(C = C_1, C_2, \dots, C_t = \widehat{C})$ be a sequence of clusters such that C_i and C_{i+1} extend into the same 2-path P_i . Consider the clustering \mathcal{C}' obtained as follows: Cluster C_1 gains one vertex from P_1 , all path clusters on P_1 are shifted by one position on the path, C_2 loses one vertex on P_1 and gains one vertex on P_2 and so on until we reach \widehat{C} which only loses one vertex on P_{t-1} .

The number of edges covered by \mathcal{C}' is the same as for \mathcal{C} . Moreover, the only two clusters whose volume has changed are C and \widehat{C} with C gaining a degree-2 vertex u and \widehat{C} losing a degree-2 vertex v . By Corollary 2.5, we have $\text{vol}(C) \leq 2|C| + 42\lambda(G)^2$ and $\text{vol}(\widehat{C} \setminus \{v\}) \geq 2(|C| + 22\lambda(G)^2 - 1) = 2|C| + 44\lambda(G)^2 - 2 > 2|C| + 42\lambda(G)^2$ since $\widehat{C} \setminus \{v\}$ is connected and $|\widehat{C} \setminus \{v\}| \geq |C| + 22\lambda(G)^2$ and $\lambda(G) \geq 2$. Thus, C , \widehat{C} , u , and v fulfill the conditions of Lemma 2.3 and \mathcal{C}' is a better clustering than \mathcal{C} . ◀

► **Lemma 3.15.** *There exists an optimal clustering \mathcal{C} such that*

- \mathcal{C} extends some full preclustering \mathcal{P} , and
- in every separated component of \mathcal{P} there is some number p such that each path cluster has size p or $p + 1$ and the base clusters have a size in the range $[p - 22\lambda^2, p + 2]$.

Proof. Note that the second statement is true for every separated component S that does not contain any path clusters, since we can set p as the size of the largest base cluster and all other base clusters in S then have a size in the range $[p - 22\lambda^2, p]$ according to Lemma 3.14. Thus it is sufficient to consider separated components that contain a path cluster.

Moreover, due to Lemma 3.7, we can assume that there is a non-empty family \mathcal{F} of optimal clusterings where in every separated component of the corresponding preclustering there is some number p such that the path clusters have size p or $p + 1$. We thus assume towards a contradiction that for every optimal clustering $\mathcal{C} \in \mathcal{F}$ and its corresponding full preclustering \mathcal{P} , there is a separated component S of \mathcal{P} that contains a base cluster C of size $|C| \notin [p - 22\lambda^2, p + 2]$, where p and $p + 1$ are the sizes of path clusters in S .

Now, let $\mathcal{C} \in \mathcal{F}$ be an optimal clustering with its corresponding full preclustering \mathcal{P} and let S be a separated component of \mathcal{P} such that in S there is a base cluster C with $|C| \notin [p - 22\lambda^2, p + 2]$. Let P be a 2-path in S that C extends into and let C_P be a path cluster in P of size p . Since $|C_P| = p > |C| + 22\lambda^2$ or $|C_P| = p < |C| - 2$, according to Lemma 3.13 the clustering \mathcal{C} is not optimal, a contradiction to the assumption. ◀

4 Solving Separated Components

We now show how to compute an optimal clustering extending a given full preclustering \mathcal{P} under the assumption that the full preclustering can be legally extended to an optimal clustering. The algorithm considers the separated components one by one. We thus assume in the following, that we are given one separated component H . Let C_1, \dots, C_t denote the base clusters of the full preclustering that are contained in H , and let P_1, \dots, P_q denote the maximal 2-paths of vertices in H that are not contained in any cluster C_i . The problem is thus to determine how far the clusters extend into the paths P_i and how large the path clusters in each 2-path P_i are.

The main observations from Section 3.2 are that for each separated component there is a number p such that the path clusters have size p or $p + 1$ and that the size of each base cluster C_i is in $[p - 22\lambda^2, p + 2]$. The algorithm to compute the optimal clustering will now consist of two main steps. First, we perform a branching to fix p and the size of each base cluster. Afterwards, we formulate the problem as an ILP.

For the branching step, first observe that the number of choices for p is less than n . Now the number of different choices for the base clusters is $\lambda^{\mathcal{O}(\lambda^2)}$ since there are $\mathcal{O}(\lambda^2)$ base clusters, and for each the number of possible sizes is $22\lambda^2 + 3$. Hence, the total number of created branches is $n \cdot \lambda^{\mathcal{O}(\lambda^2)}$.

Now, for each branch we search for an optimal clustering of H that legally extends the preclustering and fulfills all the cluster size constraints of the branch. Let c_1, \dots, c_t denote the cluster size constraints for the base clusters.

The first observation now is that the modularity of a cluster C'_i containing a cluster C_i is determined by the branch assumption: the only aspect of the cluster C_i that is not fixed by the preclustering is the total number of vertices from neighboring 2-paths of C_i that are contained in $C'_i \setminus C_i$. This number is fixed by the branching, it is precisely $c_i - |C_i|$. Each of these additional vertices contributes a value of 2 to $\text{vol}(C'_i)$ and one additional edge to $|E(C'_i)|$. Hence, the contribution of the final clusters $C'_i \supseteq C_i$ is fixed for all base clusters C_i .

Moreover, for each path cluster C the modularity contribution is

- $q_1 := (p - 1)/m - (2p)^2/4m^2$ when C is nonpendent and $|C| = p$, and
- $q_2 := p/m - (2p + 2)^2/4m^2$ when C is nonpendent and $|C| = p + 1$.
- $q'_1 := (p - 1)/m - (2p - 1)^2/4m^2$ when C is pendent and $|C| = p$, and
- $q'_2 := p/m - (2p + 1)^2/4m^2$ when C is pendent and $|C| = p + 1$.

Consequently, the only unknown quantity that influences the modularity of the clustering is the number of pendent and nonpendent path clusters that have size p and the number of pendent and nonpendent path clusters that have size $p + 1$.

With this discussion in mind, we find the optimal clustering by the following ILP. For each 2-path P_i we introduce variables $x_{1,i}$ and $x_{2,i}$ representing the number of path clusters contained in P_i of size p and $p + 1$, respectively. If P_i is pendent, then we also introduce variables $x'_{1,i}$ and $x'_{2,i}$ representing the number of pendent clusters of size p and $p + 1$, respectively. For each 2-path P_i , we declare one endpoint to be the right endpoint of P_i and one to be the left endpoint of P_i , we also introduce variables e_i^r and e_i^l that represent the number of vertices of P_i that do not belong to path clusters but to the base clusters that extend into P_i containing the right and left endpoint, respectively. Now, for a base cluster C_i , we let N_i^r denote the set of 2-paths P_j such that C_i extends from the right into P_j (that is, C_i contains the neighbor of the right endpoint of P_j) and N_i^l denote the set of 2-paths P_j such that C_i extends from the left into P_j . All variables are constrained to be nonnegative integers. Then, the ILP reads as follows.

$$\begin{aligned}
& \max \sum_{P_i} q_1 \cdot x_{1,i} + q_2 \cdot x_{2,i} + q'_1 \cdot x'_{1,i} + q_2 \cdot x'_{2,i} & (1) \\
\text{s.t.} \quad & p \cdot x_{1,i} + (p+1) \cdot x_{2,i} + e_i^r + e_i^\ell = |P_i| \quad \forall \text{ nonpendent } P_i & (2) \\
& p \cdot x_{1,i} + (p+1) \cdot x_{2,i} + p \cdot x'_{1,i} + (p+1) \cdot x'_{2,i} + e_i^r = |P_i| \quad \forall \text{ pendent } P_i & (3) \\
& \sum_{P_j \in N_i^r} e_j^r + \sum_{P_j \in N_i^\ell} e_j^\ell = c_i - |C_i| \quad \forall C_i & (4) \\
& x'_{1,i} + x'_{2,i} \leq 1 \quad \forall \text{ 1-pendent } P_i & (5) \\
& x'_{1,i} + x'_{2,i} \leq 2 \quad \forall \text{ 2-pendent } P_i & (6)
\end{aligned}$$

Here, a 1-pendent path is a pendent path with one vertex of degree 1, and a 2-pendent path is a path with two vertices of degree 1.¹

By the discussion above, the objective function (1) maximizes the modularity of the clustering for the separated component given the size constraints. Constraint (2) guarantees that the number of length- p and length- $(p+1)$ paths together with the path vertices that end up in base clusters gives the total path length for nonpendent paths. Constraint (3) guarantees the same for pendent paths. Constraint (4) guarantees that the base clusters fulfill the size constraints of the current branch. Finally, observe that Lemma 2.4 implies that the ILP has $\mathcal{O}(\lambda^2)$ variables since we have a constant number of variables for each branch of the separated component.

We now have all the necessary parts to prove the main result of this work.

► **Theorem 4.1.** *MODULARITY can be solved in $\lambda^{\mathcal{O}(\lambda^2)} \cdot n^{\mathcal{O}(1)}$ time.*

Proof. The algorithm enumerates all full preclusterings. For each full preclustering \mathcal{P} , a clustering is computed that legally extends \mathcal{P} . The correctness of the algorithm can be seen as follows. Fix an optimal clustering \mathcal{C} . Then, there is a full preclustering \mathcal{P} such that \mathcal{C} legally extends \mathcal{P} . By Lemma 3.15, there exists for each separated component of \mathcal{P} a number p such that all path clusters of the component have size p or $p+1$ and the size of each base cluster C_i is in $[p-22\lambda^2, p+2]$. For each separated component, the algorithm considers one branch where p and the sizes of the base clusters in the component are the same as the sizes of the corresponding clusters in \mathcal{C} . For this branch, the ILP computes a clustering of the component which has maximum modularity under the constraints. Thus, the modularity of the computed clustering for each separated component is the same as the modularity of \mathcal{C} for this component, and the returned clustering is globally optimal.

It remains to show the running time bound. By Lemma 3.5, the number of full preclusterings is $\lambda^{\mathcal{O}(\lambda^2)}$. For each of them, the algorithm branches for each separated component into $n \cdot \lambda^{\mathcal{O}(\lambda^2)}$ cases for the sizes of the path and base clusters. For each branch, an ILP with $\mathcal{O}(\lambda^2)$ variables is solved. This can be done in $\lambda^{\mathcal{O}(\lambda^2)} \cdot n^{\mathcal{O}(1)}$ time [6]. The overall running time follows. ◀

5 Parameterization by distance to stars

In this section, we strengthen previous hardness results for MODULARITY by showing W[1]-hardness for the parameter vertex deletion distance to disjoint union of stars. This parameter is defined as follows. Let $G = (V, E)$ be a graph. A *modulator set to a disjoint union of*

¹ These are isolated paths for which the optimal clustering can be also computed directly in polynomial time, but for the sake of brevity, we decided to describe a unified approach that can solve all separated components.

16:12 Modularity Clustering Parameterized by Max Leaf Number

stars for G is a set of vertices $S \subseteq V$, such that $G[V \setminus S]$ is a disjoint union of stars. For a graph G , the vertex deletion distance to disjoint union of stars $\text{dts}(G)$ is the size of a smallest modulator set to a disjoint union of stars for G .

We show W[1]-hardness for MODULARITY parameterized by $\text{dts}(G)$ by presenting a reduction from UNARY BIN PACKING defined as follows.

UNARY BIN PACKING

Input: A number of bins r , a capacity of a single bin k , and a multi-set of integers $A = \{a_1, \dots, a_n\}$ such that $\sum_{a \in A} a = rk$ and r and k are encoded in unary.

Question: Is there a surjective mapping $\alpha : A \rightarrow [r]$ such that for every $j \in [r]$ we have $\sum_{a \in \alpha^{-1}(j)} a = k$?

UNARY BIN PACKING is W[1]-hard for the parameter number of bins r [13]. Our reduction is an adaption of a parameterized reduction for showing the hardness of EQUITABLE CONNECTED PARTITION parameterized by the vertex deletion distance to various graph classes [2]. A main difficulty that needs to be overcome for our proof is that the sizes of the different gadgets need to be carefully balanced to achieve that a clustering corresponds to a bin packing and that a size-balanced clustering achieves the optimal modularity. In our proof, we consider the decision variant of MODULARITY where we ask if there is a clustering \mathcal{C} for G with a modularity score $q(\mathcal{C})$ (or equivalently $\tilde{q}(\mathcal{C})$) of at least some threshold value q^* .

Construction: Let $I = (A = \{a_1, \dots, a_n\}, r, k_0)$ be an instance of UNARY BIN PACKING. Let $k = k_0 \cdot r^2 \cdot n^2$ and $a_i^* = a_i \cdot r^2 \cdot n^2$. Clearly, $\sum_{i=1}^n a_i^* = r \cdot k$. Note that the instance $I^* = (A^* = \{a_1^*, \dots, a_n^*\}, r, k)$ of UNARY BIN PACKING is equivalent to I , since each item and bin size is scaled by the same factor $r^2 \cdot n^2$. We construct an instance $I' = (G, q^*)$ of MODULARITY that is equivalent to I^* as follows. Let G be an initially empty graph. For every number $a_i^* \in A^*$, we create an item gadget S_i which is a star with $a_i^* - 1$ leaf vertices and star center vertex c_i . Next, we create r bin gadgets B_1, \dots, B_r . Each of these gadgets B_j consists of a star with $p := 5r^2k^2n^2$ leaves and a star center vertex b_j . We add an edge between every center vertex b_j of a bin gadget and every center vertex c_i of an item gadget. Finally, we add $x := 8pk = 40r^2k^3n^2$ isolated edges e_1, \dots, e_x . Since $\sum_{i=1}^n a_i^* = r \cdot k$, the constructed graph G has $m := rn + rp + (rk - n) + x$ edges. This concludes the construction, except for the concrete modularity threshold q^* whose definition is deferred to the long version of this work.

Observe that after deleting all star centers b_j of bin gadgets B_j for $j \in [r]$ each connected component of the resulting graph $G' = G - (\bigcup_{j \in [r]} \{b_j\})$ is either an isolated edge e_ℓ , an item gadget S_i or an isolated vertex that was a leaf vertex of a bin gadget B_j , all of which are stars. Thus $\text{dts}(G) \leq r$ where r is the number of bins for I^* . Since UNARY BIN PACKING is W[1]-hard for the number of bins, it thus remains to show the correctness of the construction.

First, observe that, by Lemmas 2.1 and 2.2, in every optimal clustering the vertices of a star S_i belong to the same cluster. The same is true for a star B_j . Moreover, each of the isolated edges e_ℓ forms a separate cluster and we denote $\mathcal{E} := \{e_\ell \mid \ell \in [x]\}$. Thus from here on out we can assume that an optimal clustering \mathcal{C} for G has the form $\mathcal{C} = \{C_1, \dots, C_t\} \cup \mathcal{E}$, where C_i contains $r_i \in [0, r]$ bin gadgets $B_1^i, \dots, B_{r_i}^i$ as well as $n_i \in [0, n]$ item gadgets, where the item gadgets have s_i leaves in total. For the value of $\tilde{q}(\mathcal{C})$ we thus get

$$\tilde{q}(\mathcal{C}) = 4m \left(\sum_{i=1}^t |E(C_i)| \right) - \sum_{i=1}^t \text{vol}(C_i)^2 + \tilde{q}(\mathcal{E}) \quad (7)$$

$$= 4m \left(\sum_{i=1}^t r_i p + r_i n_i + s_i \right) - \sum_{i=1}^t (2r_i p + r_i n + r n_i + 2s_i)^2 + \tilde{q}(\mathcal{E}), \quad (8)$$

where $\tilde{q}(\mathcal{E})$ is the contribution of the partial clustering \mathcal{E} to the total modularity score.

The idea of the reduction is as follows. A modularity score for G of at least q^* can only be achieved by a clustering where each cluster (that is not an isolated edge e_ℓ) contains exactly one bin gadget and some item gadgets with total number of vertices k . Such a clustering corresponds to a partition of the items in A into r bins of size k_0 , scaled by the factor $r^2 \cdot n^2$. The values for p and x as well as the scaling factor $r^2 \cdot n^2$ for the items and bin sizes are chosen accordingly.

► **Theorem 5.1 (*)**. *MODULARITY is $W[1]$ -hard when parameterized by the vertex deletion distance to disjoint union of stars d_{ds} .*

6 Conclusion

We provided an FPT-algorithm for MODULARITY parameterized by a classic graph parameter, the max leaf number. Clearly, improvements of the running time for the max leaf number parameterization and FPT-algorithms for smaller structural parameters are desirable. In terms of running time improvements, it would also be interesting to reconsider and improve the FPT-algorithm for MODULARITY parameterized by the vertex cover number of G [16]. A particularly interesting question is whether one can replace the quadratic programming part for the vertex cover parameterization by a purely combinatorial algorithm or by an ILP formulation. The $W[1]$ -hardness for the parameterization by distance to stars underlines once more the algorithmic difficulty of the problem. One approach that is not ruled out by our reduction would be to combine parameterizations by vertex deletion distance to tractable graph classes with other parameterizations, for example the maximum degree of the input graph. Another approach could be to consider FPT-approximation algorithms for MODULARITY with structural parameterizations.

References

- 1 James P Bagrow. Communities and bottlenecks: Trees and treelike networks have high modularity. *Physical Review E*, 85(6):066118, 2012. doi:10.1103/PhysRevE.85.066118.
- 2 Václav Blazej, Dusan Knop, Jan Pokorný, and Simon Schierreich. Equitable connected partition and structural parameters revisited: N-fold beats lenstra. *CoRR*, abs/2404.18968, 2024. doi:10.48550/arXiv.2404.18968.
- 3 Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008. doi:10.1088/1742-5468/2008/10/P10008.
- 4 Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Görke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. On modularity clustering. *IEEE Transactions on Knowledge and Data Engineering*, 20(2):172–188, 2008. doi:10.1109/TKDE.2007.190689.
- 5 Aaron Clauset, Mark EJ Newman, and Christopher Moore. Finding community structure in very large networks. *Physical Review E*, 70(6):066111, 2004. doi:10.1103/PhysRevE.70.066111.
- 6 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
- 7 Bhaskar DasGupta and Devendra Desai. On the complexity of Newman’s community finding approach for biological and social networks. *Journal of Computer and System Sciences*, 79(1):50–67, 2013. doi:10.1016/J.JCSS.2012.04.003.
- 8 Fabien de Montgolfier, Mauricio Soto, and Laurent Viennot. Asymptotic modularity of some graph classes. In *Proceedings of the 22nd International Symposium on Algorithms and Computation (ISAAC ’11)*, volume 7074 of *Lecture Notes in Computer Science*, pages 435–444. Springer, 2011. doi:10.1007/978-3-642-25591-5_45.

16:14 Modularity Clustering Parameterized by Max Leaf Number

- 9 David Eppstein. Metric dimension parameterized by max leaf number. *Journal of Graph Algorithms and Applications*, 19(1):313–323, 2015. doi:10.7155/JGAA.00360.
- 10 Michael R. Fellows, Daniel Lokshtanov, Neeldhara Misra, Matthias Mnich, Frances A. Rosamond, and Saket Saurabh. The complexity ecology of parameters: An illustration using bounded max leaf number. *Theory of Computing Systems*, 45(4):822–848, 2009. doi:10.1007/S00224-009-9167-9.
- 11 Erez Hartuv and Ron Shamir. A clustering algorithm based on graph connectivity. *Information Processing Letters*, 76(4-6):175–181, 2000. doi:10.1016/S0020-0190(00)00142-3.
- 12 Falk Hüffner, Christian Komusiewicz, Adrian Liebtrau, and Rolf Niedermeier. Partitioning biological networks into highly connected clusters with maximum edge coverage. *IEEE ACM Transactions on Computational Biology and Bioinformatics*, 11(3):455–467, 2014. doi:10.1109/TCBB.2013.177.
- 13 Klaus Jansen, Stefan Kratsch, Dániel Marx, and Ildikó Schlotter. Bin packing with fixed number of bins revisited. *Journal of Computer and System Sciences*, 79(1):39–49, 2013. doi:10.1016/J.JCSS.2012.04.004.
- 14 Colin McDiarmid and Fiona Skerman. Modularity in random regular graphs and lattices. *Electronic Notes in Discrete Mathematics*, 43:431–437, 2013. doi:10.1016/j.endm.2013.07.063.
- 15 Colin McDiarmid and Fiona Skerman. Modularity of regular and treelike graphs. *Journal of Complex Networks*, 6(4):596–619, 2018. doi:10.1093/comnet/cnx046.
- 16 Kitty Meeks and Fiona Skerman. The parameterised complexity of computing the maximum modularity of a graph. *Algorithmica*, 82(8):2174–2199, 2020. doi:10.1007/s00453-019-00649-7.
- 17 M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69:026113, February 2004. doi:10.1103/PhysRevE.69.026113.
- 18 Mark Newman. *Networks*. Oxford university press, 2018.
- 19 Ron Shamir, Roded Sharan, and Dekel Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1-2):173–182, 2004. doi:10.1016/j.dam.2004.01.007.
- 20 Fiona Skerman. *Modularity of networks*. PhD thesis, University of Oxford, 2015.


Subset Feedback Vertex Set in Tournaments as Fast as Without the Subset

Satyabrata Jana ✉ 

University of Warwick, Coventry, UK

Lawqueen Kanesh ✉ 

Indian Institute of Technology, Jodhpur, India

Madhumita Kundu ✉ 

University of Bergen, Norway

Saket Saurabh ✉ 

The Institute of Mathematical Sciences, HBNI, Chennai, India

University of Bergen, Norway

Abstract

In the FEEDBACK VERTEX SET IN TOURNAMENTS (FVST) problem, we are given a tournament T and a positive integer k . The objective is to determine whether there exists a vertex set $X \subseteq V(T)$ of size at most k such that $T - X$ is a directed acyclic graph. This problem is known to be equivalent to the problem of hitting all directed triangles, thereby using the best-known algorithm for the 3-HITTING SET problem results in an algorithm for FVST with a running time of $2.076^k \cdot n^{\mathcal{O}(1)}$ [Wahlström, Ph.D. Thesis]. Kumar and Lokshtanov [STACS 2016] designed a more efficient algorithm with a running time of $1.6181^k \cdot n^{\mathcal{O}(1)}$. A generalization of FVST, called SUBSET-FVST, includes an additional subset $S \subseteq V(T)$ in the input. The goal for SUBSET-FVST is to find a vertex set $X \subseteq V(T)$ of size at most k such that $T - X$ contains no directed cycles that pass through any vertex in S . This generalized problem can also be represented as a 3-HITTING SET problem, leading to a running time of $2.076^k \cdot n^{\mathcal{O}(1)}$. Bai and Xiao [Theoretical Computer Science 2023] improved this and obtained an algorithm with running time $2^{k+o(k)} \cdot n^{\mathcal{O}(1)}$. In our work, we extend the algorithm of Kumar and Lokshtanov [STACS 2016] to solve SUBSET-FVST, obtaining an algorithm with a running time $\mathcal{O}(1.6181^k + n^{\mathcal{O}(1)})$, matching the running time for FVST.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Parameterized algorithms, Feedback vertex set, Tournaments, Fixed parameter tractable, Graph partitions

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.17

Funding *Satyabrata Jana*: Supported by the Engineering and Physical Sciences Research Council (EPSRC) via the project MULTIPROCESS (grant no. EP/V044621/1)

Saket Saurabh: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819416); and he also acknowledges the support of Swarnajayanti Fellowship grant DST/SJF/MSA-01/2017-18.

1 Introduction

In the d -HITTING SET problem, given a set family \mathcal{F} over a universe U of sets of size at most d and an integer k , the goal is to find a set $S \subseteq U$ of size at most k that intersects every set in \mathcal{F} . The importance of the d -HITTING SET problem stems from the number of other problems that can be re-cast in terms of it. For example, in the FEEDBACK VERTEX SET IN TOURNAMENTS (FVST) problem, the input is a tournament T together with an integer k . The task is to determine whether there exists a subset S of vertices of size at most k such that the sub-tournament $T - S$ obtained from T by removing S is acyclic. It turns out that FVST is a d -HITTING SET problem, where the vertices of T are the universe, and the family



© Satyabrata Jana, Lawqueen Kanesh, Madhumita Kundu, and Saket Saurabh; licensed under Creative Commons License CC-BY 4.0

19th International Symposium on Parameterized and Exact Computation (IPEC 2024).

Editors: Édouard Bonnet and Paweł Rzażewski; Article No. 17; pp. 17:1–17:17



Leibniz International Proceedings in Informatics

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

\mathcal{F} is the family containing the vertex set of every directed cycle on three vertices (triangle) of T . Indeed, it can easily be shown that for every vertex set S , $T - S$ is acyclic if and only if S is a hitting set for \mathcal{F} . Another example is the CLUSTER VERTEX DELETION (CVD) problem. Here, the input is a graph G and an integer k , and the task is to determine whether there exists a subset S of at most k vertices such that every connected component of $G - S$ is a clique (such graphs are called *cluster* graphs). Also, this problem can be formulated as a d -HITTING SET problem where the family \mathcal{F} contains the vertex sets of all *induced* P_3 's of G . An induced P_3 is a path on three vertices where the first and last vertex are non-adjacent in G . Other examples include SUBSET FEEDBACK VERTEX SET (SFVS) on chordal graphs, TRIANGLE PACKING IN TOURNAMENTS, INDUCED P_3 -PACKING, etc.

The best-known fixed-parameter tractable (FPT) algorithm for d -HITTING SET runs in time $\mathcal{O}^*((d - 0.7262)^k)^1$ [21]. It is also known that d -HITTING SET admits $\mathcal{O}((2d - 1)k^{d-1} + k)$ kernel [1]. This implies an algorithm with running time $\mathcal{O}^*(2.270^k)$ and a $\mathcal{O}(k^2)$ elements kernel for the 3-HITTING SET problem. However, 3-HITTING SET can also be solved in time $\mathcal{O}^*(2.076^k)$ [23]. For a long time, advancements in its kernel size and FPT algorithms have stagnated. In response, researchers have shifted focus towards designing algorithms and kernels for implicit 3-HITTING SET problems such as FVST, CVD.

The design of algorithms with a running time better than $\mathcal{O}^*(2^k)$ and kernels with subquadratic elements for implicit 3-HITTING SET problems has emerged as a highly active and significant area of research. In pursuit of this objective, several important problems have been investigated, leading to notable successes. Fomin et al. in [9] broke the kernel barrier of some of the implicit 3-HITTING SET problems and obtained subquadratic kernels for several problems such as $\mathcal{O}(k^{3/2})$ vertex kernel for FVST, $\mathcal{O}(k^{5/3})$ vertex kernel for CVD, $\mathcal{O}(k^{3/2})$ vertex kernel for TRIANGLE PACKING IN TOURNAMENTS (TPT), and $\mathcal{O}(k^{5/3})$ vertex kernel for INDUCED P_3 -PACKING. Recently, Bessy et al. [3] introduced a novel technique called rainbow matching to design kernels for implicit 3-HITTING SET problems. They demonstrated that TPT and FVST admit (almost linear) kernels of $\mathcal{O}(k^{1 + \frac{\mathcal{O}(1)}{\sqrt{\log k}}})$ vertices. Utilizing the same technique, they showed that INDUCED 2-PATH-PACKING and INDUCED 2-PATH HITTING SET admit kernels of $\mathcal{O}(k)$ vertices.

Another well-studied implicit 3-HITTING SET problem is SUBSET FEEDBACK VERTEX SET (SFVS) on chordal and split graphs. A feedback vertex set in a graph G is a vertex set whose removal makes the remaining graph acyclic. The FEEDBACK VERTEX SET problem (FVS) is to decide whether a graph has a feedback vertex set of size at most k . In the more general SFVS problem, an additional subset S of vertices is given, and we want to find a vertex set of size at most k that hits all cycles passing through a vertex in S . SFVS has been shown to admit an algorithm with a running time of $\mathcal{O}^*(4^k)$ [15, 16]. However, on chordal and split graphs, Philip et al. [22] showed that this problem could be solved in $\mathcal{O}^*(2^k)$ time.

In this paper, we focus on SUBSET FEEDBACK VERTEX SET IN TOURNAMENTS (SUBSET-FVST). Below, we formally define the problem.

SUBSET FEEDBACK VERTEX SET IN TOURNAMENTS (SUBSET-FVST) **Parameter:** k
Input: A tournament $T = (V, A)$, a vertex set $S \subseteq V$, and a positive integer k .
Task: Find $X \subseteq V$ with $|X| \leq k$ such that $T - X$ has no cycle containing a vertex of S .

A tournament is a directed graph formed by a complete graph with oriented arcs. Motivated by applications such as voting systems and rank aggregation, FVST has drawn certain interests. It is well known that a tournament has a directed cycle if and only if there

¹ We use \mathcal{O}^* notation to hide factors polynomial in the input size.

is a directed triangle [7]. The fastest algorithm for FVST runs in time $\mathcal{O}^*(1.619^k)$ [19] and by the rainbow matching technique of Bessy et al. [3] FVST admit (almost linear) kernel of $\mathcal{O}(k^{1+\frac{\mathcal{O}(1)}{\sqrt{\log k}}})$. SUBSET-FVST can also be framed as a 3-HITTING SET problem, resulting in an algorithm with running time $\mathcal{O}^*(2.076^k)$ and an $\mathcal{O}(k^2)$ kernel. Recently, in 2023, Bai and Xiao [2] improved this and obtained an algorithm that runs in time $\mathcal{O}^*(2^{k+o(k)})$; however, the question to obtain an algorithm with running time better than $\mathcal{O}^*(2^k)$ was left open. In our research, we build upon the algorithm of Kumar and Lokshtanov [19] to tackle SUBSET-FVST, achieving an algorithm with running time $\mathcal{O}^*(1.6181^k)$, that matches the running time for FVST. We obtain the following theorem.

► **Theorem 1.** SUBSET-FVST is solvable in $\mathcal{O}(1.6181^k + n^{\mathcal{O}(1)})$ time.

Ideas for Theorem 1. We closely follow the approach of Kumar and Lokshtanov [19], but due to the inherent generality of our problem, we need to deviate significantly from it while implementing the outline. Let (T, S, k) be an instance of SUBSET-FVST. The algorithm relies on a simple observation that T has no S -cycle (directed cycle containing a S vertex) if and only if T has no S -triangle. Our algorithm enumerates subexponential many sets ($2^{o(k)}$) or branches with a branching vector $(1, 2)$. The algorithm first identifies $2^{o(k)}$ subsets of S , such that for every solution H , there is at least one set, say M , that is disjoint from it. The set M allows us to discover several structures and apply reduction rules. For example, we know that $T[M]$ is a directed acyclic graph (DAG). In other words, if any vertex $v \in V(T) - M$, we have that $T[M \cup \{v\}]$ has a directed cycle, then v must be in H . Let σ be a unique topological ordering of $T[M]$. This immediately gives us the notion of M -block: the set of vertices which are common out-neighbors and in-neighbors of two consecutive vertices of M in σ (and not containing any M vertex). Now we analyze S -triangles: those that are fully contained inside a block (local triangle) or contain vertices of at least 2 blocks (shared triangle). Our main objective is to reduce the case of shared triangles to a vertex-cover like branching (either a vertex or its neighbors must be in a solution) and independently solve the problem of hitting local triangles. In the latter case, we use the fact that each block has at most $\log^{\mathcal{O}(1)} k$ many vertices from S and hence any solution must contain at most $\log^{\mathcal{O}(1)} k$ many vertices from each block. Thus, using the fact that SUBSET-FVST has a polynomial kernel of size $\mathcal{O}(k^2)$, we can solve these instances in $k^{\mathcal{O}(\log^{\mathcal{O}(1)} k)}$ time. The most interesting part of the algorithm is to reduce to this case. This requires branching on “backward arcs” between two blocks. If there is a vertex v with at least two incident backward arcs, then we branch on v , leading to a branching vector $(1, 2)$. When this is not possible, then we have that these backward arcs form a matching. In this case if no block has many backward arcs incident then we can partition these edges and decompose the problem. This leads to a divide-and-conquer step in our algorithm. This concludes a brief description of our algorithm.

Related Work on Feedback Vertex Set

The FEEDBACK VERTEX SET problem (FVS) is one of the earliest known NP-complete problems shown in the influential paper by Karp [18] and has been thoroughly explored in the realm of parameterized complexity. The earliest known FPT algorithms for FVS date back to the late 1980s and early 1990s. These algorithms relied on the groundbreaking Graph Minor Theory by Robertson and Seymour. Over time, there have been multiple advancements and refinements, leading to the current best deterministic FPT algorithm for FVS, which runs in $\mathcal{O}^*(3.460^k)$ time [14]. The fastest known randomized algorithm for this problem given by Li and Nederlof [20] running in time $\mathcal{O}^*(2.7^k)$. Recently a factor

$(1 + \epsilon)$ approximation algorithm for FVS, which has better running time than the best-known (randomized) FPT algorithm for every $\epsilon \in (0, 1)$ is given by Jana et al. [17]. In directed graphs, FVS becomes harder in terms of parameterized algorithms. Whether FVS in directed graphs is FPT has been a long-standing open problem. Finally, Chen et al. [4] gave a FPT algorithm running in time $\mathcal{O}^*(4^k k!)$.

SUBSET FEEDBACK VERTEX SET problem (SFVS) was first systematically studied by Even et al. [8] where they showed that SFVS in undirected graphs admits an 8-approximation. Cygan et al. [6] showed that SFVS in undirected graphs admits an FPT algorithm running in time $\mathcal{O}^*(2^{\mathcal{O}(k \log k)})$. Recently, Iwata et al. [15, 16] improved this to single-exponential algorithm with running time $\mathcal{O}^*(4^k)$. This problem is also studied in special graph classes. Philip et al. [22] showed that in split and chordal graphs this problem can be solved in time $\mathcal{O}^*(2^k)$. Regarding kernelization, Hols and Kratsch [13] obtained a randomized kernel with $\mathcal{O}(k^9)$ vertices. Chitnis et al. [5] considered this problem in directed graph and provide a FPT algorithm running in time giving a $\mathcal{O}^*(2^{\mathcal{O}(k^3)})$.

2 Preliminaries

Let $[n]$ be the set of integers $\{1, \dots, n\}$. For a pair a, b of integers with $a < b$, we denote the set $\{a, a + 1, \dots, b\}$ by $[a, b]$. For a directed graph D , we denote the set of vertices of D by $V(D)$ and the set of arcs by $A(D)$. For a subset X of vertices $X \subseteq V(D)$, we use the notation $D - X$ to mean the graph $D[V(D) \setminus X]$. Given a digraph D , a vertex set $X \subseteq V(D)$ is called a *feedback vertex set* (in short, fvs) of D if there is no directed cycle in the graph $D - X$. Given a directed graph D and a vertex set S , a vertex $v \in V(D)$ is called an *S-vertex* if $v \in S$. A directed cycle in D is called an *S-cycle* if the cycle contains at least one *S-vertex*. A *S-cycle* is called an *S-triangle* if it is a cycle of three vertices. D is called *S-acyclic* if D has no *S-cycle*. A vertex set $X \subseteq V(D)$ is called an *S-feedback vertex set* of D (in short, *S-fvs*) if $D - X$ is *S-acyclic*. Let σ be an ordering of $V(D)$. For a pair of adjacent vertices u, v with an arc (u, v) , we say the arc is *backward* (resp, *forward*) with respect to the ordering σ if $v \leq_\sigma u$ (resp, $u \leq_\sigma v$). It is called an *S-backward* arc (resp, *S-forward* arc) if there exists some *S-vertex* s such that $v \leq_\sigma s \leq_\sigma u$ (resp, $u \leq_\sigma s \leq_\sigma v$). We call an ordering without *S-backward* arcs an *S-topological ordering*. A directed graph is called a *tournament* if there is an arc between every pair of vertices. Unless specified, we use cycle to mean directed cycle.

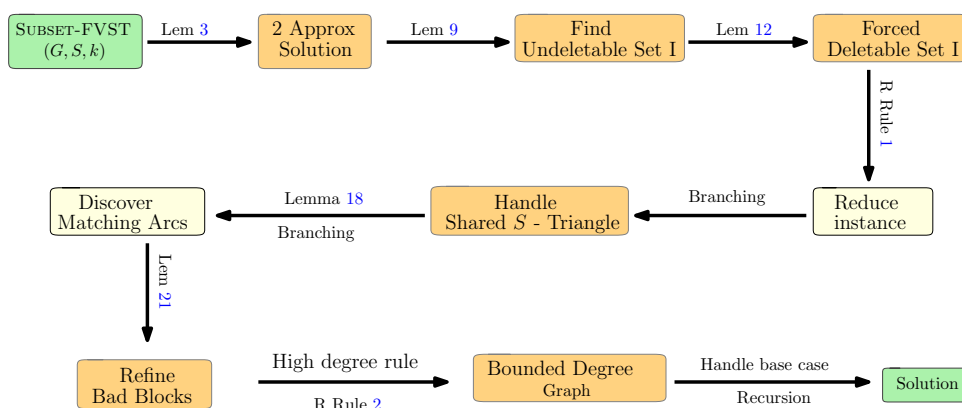
Fixed parameter tractable. A parameterized problem Π is a subset of $\Gamma^* \times \mathbb{N}$ for some finite alphabet Γ . An instance of a parameterized problem consists of (X, k) , where k is called the parameter. A parameterized problem L is considered to have a fixed parameter tractable (FPT) algorithm if there is an algorithm \mathcal{A} that can determine whether $(X, k) \in L$ in time $f(k) \cdot n^{\mathcal{O}(1)}$ for some computable function f , where n is the size of the input. An important tool from the FPT toolkit is *kernelization* [10, 11]. A kernelization replaces, in polynomial time, an instance by a decision equivalent instance (the kernel) whose size can be bounded by a function of the parameter k , that is, it will not depend on the original problem size n anymore.

3 FPT algorithm for SUBSET-FVST

In this section, we prove the following result.

► **Theorem 1.** SUBSET-FVST is solvable in $\mathcal{O}(1.6181^k + n^{\mathcal{O}(1)})$ time.

A schematic diagram showing the main steps of our algorithm is shown in Figure 1.



■ **Figure 1** A summary of the steps of our algorithm.

3.1 Preprocessing Step

It is well-known that a tournament is acyclic if and only if it does not contain any triangle [7], which allows us to formulate FVST as a 3-HITTING SET problem. We first observe a similar statement for the subset variant in the next lemma proved in [2].

► **Lemma 2** ([2, Lemma 2]). *A tournament is S -acyclic if and only if it does not contain an S -triangle.*

Lemma 2 immediately gives rise to a greedy 3-approximation algorithm for SUBSET-FVST. However, Gupta et al. [12] designed a 2-factor approximation algorithm for SUBSET-FVST. Using this result, we get the following lemma.

► **Lemma 3.** *Given a tournament T with n vertices, a subset $S \subseteq V(T)$ and integer k , in $n^{\mathcal{O}(1)}$ time we correctly conclude that T has no S -feedback vertex set of size at most k or outputs a S -feedback vertex set of size at most $2k$.*

The initial phase of our algorithm for SUBSET-FVST involves reducing the problem to its kernel. By reducing the instance of the SUBSET-FVST problem into an instance of the 3-HITTING SET problem, by using a kernel for 3-HITTING SET [1], one can derive a kernel on $\mathcal{O}(k^2)$ vertices for SUBSET-FVST, which is an induced subgraph of the input tournament. Formally, we have the following lemma.

► **Lemma 4** ([1, 2]). *Given a tournament T with n vertices, a subset $S \subseteq V(T)$ and an integer k , in $n^{\mathcal{O}(1)}$ time we can output a tournament T' (an induced subgraph), a vertex set $S' \subseteq V(T')$ and an integer k' such that $|V(T')| \leq \mathcal{O}(k^2)$, $k' \leq k$, and T' has a S' -feedback vertex set of size at most k' if and only if T has a S -feedback vertex set of size at most k .*

In what follows, we assume that we have applied Lemma 4 and obtained an equivalent instance, (T, S, k) , such that $|V(T)| \leq \mathcal{O}(k^2)$. We call such instances *reduced*.

3.2 Discovering Structure I: Universal Undeletable Family \mathcal{M}

In the second step of our algorithm, we find a family of subsets of S such that any solution avoids at least one set in our family. Toward defining the family we first need a notion of block which relies on the notion of between and consecutive.

17:6 Subset FVS in Tournaments as Fast as Without the Subset

- **Definition 5** (Between and Consecutive). Let D be a directed graph.
- For any pair of vertices $u, v \in V(D)$, the set $\text{between}(D; u, v)$ is defined as $N^+(u) \cap N^-(v) \setminus \{u, v\}$.
 - Let $X \subseteq V(D)$. Two vertices $u, v \in X$ are called X -consecutive if $(u, v) \in A(D)$ and $\text{between}(D; u, v) \cap X = \emptyset$.

► **Definition 6** (X -block). Let D be a directed graph and $X \subseteq V(D)$. We define the set of X -blocks in D as follows.

- For each pair of X -consecutive vertices u and v , we define the X -block, denoted by $\text{block}(X; u, v)$, as $\text{between}(D; u, v)$. That is, $\text{block}(X; u, v) = \text{between}(D; u, v)$.
- For each vertex $u \in X$ with no in-neighbors in X we define the X -block, denoted by $\text{block}(X; u)$, as $N^-(u)$. That is, $\text{block}(X; u) = N^-(u)$.
- For each vertex $v \in X$ with no out-neighbors in X we define the X -block, denoted by $\text{block}(X; v)$, as $N^+(v)$. That is, $\text{block}(X; v) = N^+(v)$.

These definitions immediately imply the following observation for an acyclic tournament.

► **Observation 7.** Let T be an acyclic tournament and $X \subseteq V(T)$. Furthermore, let σ be the unique topological order of $V(T)$. Then the following holds.

1. For any pair of vertices $u, v \in V(T)$ the set $\text{between}(T; u, v)$ is exactly the set of vertices in T that appear between u and v in σ .
2. Two vertices $u, v \in X$ are X -consecutive if no vertex of X appears between u and v in σ .
3. X -blocks form a unique partition of $V - X$, where two vertices belong to a different block if and only if there exists a vertex of X that appears between them in σ .

Now we are ready to define the notion of universal undeletable family.

► **Definition 8** (Universal Undeletable Family). Let (T, S, k) be an instance of SUBSET-FVST. A *universal undeletable family*, \mathcal{M} , is a family of subsets of S , which satisfies the following properties. For every S -feedback vertex set H of T size at most k there exists a set $M \in \mathcal{M}$ such that

1. $H \cap M = \emptyset$;
2. in each M -block B in $T - H$ we have $|B \cap S| \leq 2 \log^2 k$.

Our main result in this section is the following.

► **Lemma 9.** Let (T, S, k) be a reduced instance. There exists an algorithm that takes (T, S, k) as input and outputs a universal undeletable family \mathcal{M} of size $2^{\mathcal{O}(\frac{k}{\log k})}$.

Proof. Let X be a S -feedback vertex set of size at most $2k$ obtained using Lemma 3. Let $Y := V(T) \setminus X$, $S_X := X \cap S$, $S_Y := Y \cap S$ and $v_1, v_2, \dots, v_{|S_Y|}$ be the topological sort of $T[S_Y]$ such that the edges in $T[S_Y]$ are directed from left to right. We color S_Y using $\lfloor \log^2 k \rfloor$ colors $\{1, 2, \dots, \lfloor \log^2 k \rfloor\}$ such that for each $d \in [|S_Y|]$, v_d gets color $d \bmod \lfloor \log^2 k \rfloor$. For each $c \in [\lfloor \log^2 k \rfloor]$, let S_c be the set of vertices in S_Y which get color c . Each $M \in \mathcal{M}$ is specified by a 4-tuple $\langle c, \hat{H}, \hat{R}, \hat{X} \rangle$ where

- c is a color in $[\lfloor \log^2 k \rfloor]$,
- $\hat{H} \subseteq S_c$ such that $|\hat{H}| \leq \frac{k}{\log^2 k}$,
- $\hat{R} \subseteq S_Y \setminus S_c$, $|\hat{R}| \leq |\hat{H}|$, and
- $\hat{X} \subseteq X$ such that $|\hat{X}| \leq \frac{2k}{\log^2 k}$.

For each 4-tuple $\langle c, \hat{H}, \hat{R}, \hat{X} \rangle$, let $M := (S_c \setminus \hat{H}) \cup \hat{R} \cup \hat{X}$. Hence $|\mathcal{M}|$ is upper bounded by the maximum possible number of such 4-tuples.

$$\begin{aligned}
|\mathcal{M}| &\leq \log^2 k \times \left(\frac{\mathcal{O}(k^2)}{\frac{k}{\log^2 k}}\right) \times \left(\frac{\mathcal{O}(k^2)}{\frac{k}{\log^2 k}}\right) \times \left(\frac{2k}{\frac{2k}{\log^2 k}}\right) \\
&\leq 2^{\log(\log^2 k)} \times \mathcal{O}(k^2)^{\frac{2k}{\log^2 k}} \times (2k)^{\frac{2k}{\log^2 k}} \\
&= 2^{\mathcal{O}(\frac{k}{\log k})}
\end{aligned}$$

This completes the description of the enumeration of \mathcal{M} . Next, we prove the correctness of the above algorithm by showing that for every S -feedback vertex set H of T with size at most k , \mathcal{M} contains a set M that satisfies the properties listed in the statement of the lemma. Consider an arbitrary S -feedback vertex set H of T with size at most k .

For each $j \in [\lceil \log^2 k \rceil]$, let $H_j := S_j \cap H$. By the pigeonhole principle, there is a color c such that $0 < |H_j| \leq \frac{k}{\log^2 k}$. For this color c , let $\hat{H} := H_c$. Note that $H_c \subseteq S_c \subseteq S_Y$. Now, consider a set \hat{R} obtained as follows: for every vertex $v \in H_c$, pick the first vertex after v (if there is any) in $S_Y \setminus (S_c \cup H)$, in the topological ordering of $T[S_Y]$. Note that $T[X \setminus H]$ is S -acyclic. Let $S_X^H := S \cap (X \setminus H)$. Now $|S_X^H| \leq 2k$ and there is a topological ordering of $T[S_X^H]$. We then color S_X^H using $\lceil \log^2 k \rceil$ colors in a similar way as we did for S_Y . Let $\hat{X} \subseteq S_X^H$ be the set of all vertices with color 1 in this coloring. Clearly, $|\hat{X}| \leq \frac{2k}{\log^2 k}$.

The 4-tuple $\langle c, \hat{H}, \hat{R}, \hat{X} \rangle$ described above satisfies all the properties listed in the construction of \mathcal{M} . Let $M := (S_c \setminus \hat{H}) \cup \hat{R} \cup \hat{X}$. Clearly, $M \subseteq S$, $M \cap H = \emptyset$, and $M \in \mathcal{M}$. Since in any $[(S_c \setminus H_c) \cup \hat{R}]$ -block B in Y , it holds $|B \cap S| \leq \log^2 k$, it also holds in each M -block B in $T - H$ that $|B \cap S| \leq 2 \log^2 k$. \blacktriangleleft

3.3 Discovering Structure II: Forced Deletable Set I

Lemma 9 gets us one step closer to our goal. Let H be a hypothetical solution of size at most k of (T, S, k) . We know that there exists a set $M \in \mathcal{M}$ such that in each M -block in $T - H$, we have a small number of S -vertices, i.e., each M -block contains a small number ($\leq 2 \log^2 k$) of non-solution S -vertices. However, it is possible that there is a M -block of T that contains too many vertices of S , because that block contains *many solution S -vertices, that is, vertices of $H \cap S$* . In this section, we deal with this case. We assume that we know M corresponding to the hypothetical solution H . Indeed, we can achieve this by going over each set in \mathcal{M} , and that will only cost us a factor of $2^{\mathcal{O}(\frac{k}{\log k})}$ in the running time of our algorithm. It will be useful to remember that $T[M]$ is a directed acyclic tournament.

We start with a simple definition that will be useful.

► Definition 10 (Consistency). Let T be a tournament and $M \subseteq V(T)$. For a vertex $v \in V(T)$, we say that v is *consistent* with M if there is no $\{v\}$ -cycle in $T[M \cup v]$.

Observe that if v is not consistent with M , then there is a v -cycle in $T[M \cup v]$, which is also a S -cycle, since $M \subseteq S$. Thus, since we know that $M \cap H = \emptyset$, v must belong to H . We next integrate such vertices and blocks that may contain a lot of S vertices (in the solution) into the notion of universal undeletable families. We call this notion universal (deletable-undeletable) families (in short, *du-family*).

► Definition 11 (du-Family). Let (T, S, k) be an instance of SUBSET-FVST. A *du family*, $\mathcal{F} = \{(M_1, P_1), \dots, (M_\ell, P_\ell)\}$, is a family of disjoint pairs of vertex sets, which satisfies the following properties. For every S -feedback vertex set H of T size at most k there exists a set pair $(M, P) \in \mathcal{F}$ such that the following holds.

1. $M \subseteq S$, $M \cap H = \emptyset$.
2. $P \subseteq H$ (recall, $M \cap P = \emptyset$).

3. Every vertex of $T - P$ is consistent with M .
 4. In each M -block B in $T - P$, we have $|B \cap S| \leq 2 \log^4 k$.
- Furthermore, the set H is said to be *compatible* with the pair (M, P) .

► **Lemma 12.** *Let (T, S, k) be a reduced instance. There exists an algorithm that takes (T, S, k) as input and outputs a du-family \mathcal{F} of size $2^{\mathcal{O}(\frac{k}{\log k})}$.*

Proof. We define a function f that for a given tournament T and a subset $M \subseteq V(T)$, outputs a set of vertices that are *not consistent* with M . We denote this output set as $f(T, M)$. More specifically

$$f(T, M) := \{v \mid v \in V(T) \setminus M \text{ and } T[M \cup \{v\}] \text{ has a } \{v\}\text{-cycle}\}$$

Observe that each v -cycle here is also a S -cycle since $M \subseteq S$. Thus, since we know that $M \cap H = \emptyset$, v must belong to H . Furthermore, we define another function g that, given a tournament T , two subsets $M \subseteq S \subseteq V(T)$, and an integer k , outputs the S -vertices contained within some M -block B in $T - f(T, M)$ with $|B \cap S| > 2 \log^4 k$. We denote this output set as $g(T, S, M, k)$. We can compute f and g in time $k^{\mathcal{O}(1)}$.

We use the algorithm of Lemma 9 to compute \mathcal{M} . Then for each $M \in \mathcal{M}$, we compute the sets $f(T, M)$ and $g(T, S, M, k)$. For each $Z \subseteq g(T, S, M, k)$ such that $|Z| \leq \frac{2k}{\log^2 k}$, we output a pair of sets $(M, P) = (M, f(T, M) \cup (g(T, S, M, k) \setminus Z))$. The family \mathcal{F} is the collection of all such pairs of sets.

We prove that the algorithm satisfies the stated properties. Consider a S -feedback vertex set H of size at most k . By Lemma 9 there exists $M \in \mathcal{M}$ such that $M \subseteq S$ and $M \cap H = \emptyset$. Because of the definition of $f(T, M)$ and $M \cap H = \emptyset$, the vertex set $f(T, M)$ must be contained in H . Now for every vertex $u \in V(T - M) \setminus f(T, M)$ the induced subtournament $T[M \cup \{u\}]$ of T is acyclic. So u can be placed uniquely in the topological order of $T[M \cup \{u\}]$. Hence, for each $u \in V(T - M) \setminus f(T, M)$, there is a unique M -block containing it. Since in each M -block B in $T - H$ we have $|B \cap S| \leq 2 \log^2 k$, we also have in each M -block B in $T - f(T, M)$ that $|B \cap S| \leq k + 2 \log^2 k$.

We say that a M -block B in $T - C$ is *large* if $|B \cap S| \geq 2 \log^4 k$. From each large M -block at least $(2 \log^4 k - 2 \log^2 k)$ many S -vertices belong to H . So the number of large blocks is bounded by $\frac{k}{2 \log^4 k - 2 \log^2 k}$. Hence in total at most $\frac{k}{2 \log^4 k - 2 \log^2 k} \times 2 \log^2 k \leq \frac{2k}{\log^2 k}$ many S -vertices from the union of large M -blocks do not belong to H . Since the algorithm loops over all choices of subsets $Z \subseteq g(T, S, M, k)$ with $|Z| \leq \frac{2k}{\log^2 k}$, the family \mathcal{F} contains a pair (M, P) satisfying the properties listed in the lemma.

Moreover, $|\mathcal{F}|$ is bounded by the product of $|\mathcal{M}|$ and the number of subsets Z . Now $Z \subseteq S$ and $|Z| \leq \frac{2k}{\log^2 k}$ together imply that the number of subsets Z is at most $\mathcal{O}(k^2)^{\frac{2k}{\log^2 k}} = 2^{\mathcal{O}(\frac{k}{\log k})}$. Hence $|\mathcal{F}| = |\mathcal{M}| \times 2^{\mathcal{O}(\frac{k}{\log k})} \leq 2^{\mathcal{O}(\frac{k}{\log k})} \times 2^{\mathcal{O}(\frac{k}{\log k})} = 2^{\mathcal{O}(\frac{k}{\log k})}$. ◀

Our algorithm for SUBSET-FVST applies Lemma 12 and obtains a du-family \mathcal{F} . The algorithm then iterates over each pair $(M, P) \in \mathcal{F}$, and looks for a S -feedback vertex set H of size at most k that is compatible with (M, P) . Henceforth, our problem reduces to the following.

Given an instance (T, S, k) , and a pair (M, P) of vertex sets in T . The objective is to find a S -feedback vertex set H for T of size at most k that is compatible with (M, P) .

We first apply the following reduction rule to make the instance smaller. The correctness of Reduction Rule 1 follows from the fact that we are looking for a S -feedback vertex set H for T of size at most k that is compatible with (M, P) (Lemma 12).

► **Reduction Rule 1.** Delete P from the graph. The resultant instance is $(T - P, S \setminus P, k - |P|)$.

The pair (M, P) naturally partitions the vertices of $T - (P \cup M)$ into local subtournaments corresponding to the induced graphs on the M -blocks in $T - P$. For clarity of notation, we denote the reduced instance $(T - P, S \setminus P, k - |P|)$ by (T, S, k) itself. The properties of $(T - P, S \setminus P, k - |P|) = (T, S, k)$ that we need in the future are encapsulated below.

1. For all $v \in V(T) \setminus M$, we have that $T[M \cup \{v\}]$ does not contain a $\{v\}$ -cycle. In particular, $T[M \cup \{v\}]$ is acyclic.
2. In each M -block B in T , we have $|B \cap S| \leq 2 \log^4 k$.

Following the implementation of Reduction Rule 1, we categorize S triangles in T into two distinct groups, as described below.

- (i) **Local S -triangle:** All three vertices are within one M -block in T .
- (ii) **Shared S -triangle:** Those that are not local.

Notice that in each shared S -triangle (not containing a vertex of M) there exists a pair of vertices in the triangle that belong to different M -blocks in $T - P$. Next, we understand how M participates in a S -triangle. Observe that every S -triangle contains at most one vertex from M . Furthermore, we show that every S -triangle containing a M vertex (which we say M -triangle) must be a shared S -triangle.

► **Observation 13.** Every M -triangle is a shared S -triangle. That is, there is no pair of vertices u, v such that both vertices u, v belong to the same M -block in T and there is a vertex $w \in M$ that forms a triangle with u, v .

Proof. Let u, v be a pair of vertices in T . Since we have applied Reduction Rule 1, it follows that the vertex $u \in V(T)$ is consistent with M , meaning that there is no $\{u\}$ -cycle in $T[M \cup \{u\}]$. That implies $T[M \cup \{u\}]$ is a DAG. Thus, there exists a unique partition $M_1 \cup M_2$ of M such that for all $x \in M_1, y \in M_2$ we have $(x, u), (u, y) \in V(T)$. Similarly, since $T[M \cup \{v\}]$ is also a DAG, there exists a unique partition $M'_1 \cup M'_2$ of M such that for all $x' \in M'_1, y' \in M'_2$ we have $(x', v), (v, y') \in V(T)$. Furthermore, since u and v belong to the same M -block, it must be that $M_1 = M'_1$ and $M_2 = M'_2$. Therefore, both $T[M_1 \cup \{u, v\}]$ and $T[M_2 \cup \{u, v\}]$ are DAGs. Now, the vertex w is in either M_1 or M_2 . In either case, w cannot form a cycle with $\{u, v\}$. This concludes the proof. ◀

Next, we show that having shared S -triangle is equivalent to having a M -triangle

► **Lemma 14.** If there is a shared S -triangle uvw , then there exists a vertex $m \in M$ such that there is a shared S -triangle umw .

Proof. If there is a shared S -triangle, say uvw , then it must contain vertices from two different M blocks, say B_1 and B_2 , where the vertices in B_1 appear before B_2 and there is an arc (v, u) with $u \in B_1$ and $v \in B_2$. Let m be a vertex of M that appears before any vertex of B_2 and after every vertex of B_1 . This implies we have arcs (u, m) and (m, v) . This implies that we have a triangle umv containing a M vertex. ◀

Observation 13 and Lemma 14 imply the following.

► **Lemma 15.** There is a shared S -triangle if and only if there is a M -triangle.

17:10 Subset FVS in Tournaments as Fast as Without the Subset

Before we proceed further, we design an algorithm for SUBSET-FVST that runs in time $2^{k+o(k)} + n^{\mathcal{O}(1)}$. Branch on each of the triangles containing M -vertices. This will lead to 2^k branching factor. Lemma 15 implies that the only S -triangle that remains after branching are local S -triangles, and hence we can solve the problem independently on each block. However, observe that in each M -block B in T , we have $|B \cap S| \leq 2 \log^4 k$. This implies that the size of an optimal solution in B is upper-bounded by $2 \log^4 k$. Thus, we can solve the problem for each block independently in time $k^{\mathcal{O}(\log^4 k)}$. This results in the following.

► **Theorem 16.** SUBSET-FVST is solvable in time $2^{k+o(k)} + n^{\mathcal{O}(1)}$.

This algorithm is comparable to the best known algorithm of Bai and Xiao [2]. We move on to designing the faster algorithm. This algorithm also follows the route of Theorem 1. We move on to Section 3.4 when there is at least one shared S -triangle, otherwise we move to Section 3.7.

3.4 Branching Structure I: Shared S -triangle

Our objective is to reduce to “vertex cover” (either a vertex or all of its neighbors must go in any solution) like branching in this case. Towards this, we first define the set of edges for which we need vertex cover.

Dealing with shared S -triangles. Consider the ordered partition of the vertices of T based on M vertices in the following way: each M vertex is a singleton and they are ordered according to the unique topological ordering σ of $T[M]$. The other parts are defined by M -block with corresponding order. For example, let $|M| = \ell$ and m_1, \dots, m_ℓ be the vertices of M such that $m_1 < \dots < m_\ell$ in σ . Then our ordered partition of T would be

$$B_1 = \text{block}(X, m_1) \cup \{m_1\} \cup B_2 = \text{block}(M; m_1, m_2) \cup \dots \cup \{m_\ell\} \cup B_{\ell+1} = \text{block}(X, m_\ell).$$

Observe that every vertex v not in M belongs to a unique block as $T[M \cup \{v\}]$ is acyclic.

Let R be the set of arcs (u, v) of $A(T)$ such that $u \in B_j$ and $v \in B_i$ and $i, j \in [\ell + 1]$, and $i < j$. We call R a set of *red arcs*. Next, we show that dealing with shared triangles is equivalent to hitting arcs in R .

► **Lemma 17.** *Every shared S -triangle has a red arc. Furthermore, for every red arc $e = (u, v) \in R$ we have a M -triangle containing e . This implies that we have a shared S -triangle if and only if there is a red arc.*

Proof. If a S -triangle is not local, then it contains vertices of two distinct blocks or two vertices of the same block and a M vertex. By Observation 13 we know that there is no S -triangle containing two vertices of the same block and a M vertex. It is clear that any S -triangle containing vertices of two distinct blocks contains a red arc.

Let $e = (u, v) \in R$. Then there exists $i, j \in [\ell + 1]$ such that $u \in B_j$ and $v \in B_i$ where $i < j$. Let m be a vertex of M that appears before any vertex of B_j and after every vertex of B_i . We know from the properties of the M blocks that we have arcs (v, m) and (m, u) . This implies that we have a triangle vmu containing a M vertex. This concludes the proof. ◀

► **Lemma 18.** *Let H be a solution to (T, S, k) . Then H is a vertex cover of R (that is, it intersects each edge in R).*

Proof. For every red arc $e = (u, v) \in R$ we have a M -triangle containing e (Lemma 17). Since, by construction no red arc has an end-point in M and $M \cap H = \emptyset$ by assumption, we have that H must contain either u or v . This concludes the proof. ◀

We first consider the case where there are some vertices that are incident to more than one red arc. Once we deal with that case, the only situation left is that all the red arcs that remain to hit form a matching. Now we describe a recursive algorithm which searches for a potential solution H of size at most k by branching. Let $G = (V(T), R)$ be an undirected graph with vertex set $V(T)$ and edge set R (without orientation). For any vertex v , let $\text{Red}(v)$ (the *red-degree*) denote the degree of v in G . Consider a vertex v of highest red-degree. We know $\text{Red}(v) > 0$. If $k = 0$, then return No. From now on, assume that $k \geq 1$. If $\text{Red}(v) \geq 2$, the algorithm branches into two cases: $v \in H$ or $v \notin H$. In the branch where v is added to H , k drops by 1. Hence, we return the instance $(T - v, S \setminus \{v\}, k - 1)$. In the other branch where v is not added to H , we know that all the neighbors in G must be in the solution (by Lemma 18). Hence, we return the instance $(T - N_G(v), S \setminus N_G(v), k - \text{Red}(v))$. Here, $N_G(v)$ denotes the neighbors of v in G . This (1, 2) branching step dominates the running time of the algorithm and corresponds to $\mathcal{O}(1.6181^k \times k^{\mathcal{O}(1)})$ time.

Now we assume that maximum red-degree of any vertex is at most 1. In this case, all the red arcs form a matching in G . We refer to these kind of red arcs as *matching arcs*. Let (T, S, k, M) obtained after branching be called the *matching instance*. In Section 3.5, we look for more structure in the input and take advantage of it.

3.5 Discovering Structure III: Matching Arcs

We now introduce the notion of a *bad* block to represent blocks that are incident to a *large* number of matching arcs.

► **Definition 19** (Bad block). An M -block B is said to be *bad* if the number of matching arcs incident to the vertices in B is at least $\log^4 k$. Otherwise we say that the block B is *good*.

Now, in the following observation, we show that the number of bad blocks will be bounded.

► **Observation 20.** *The number of bad M -blocks in T is at most $\frac{2k}{\log^4 k}$.*

Proof. We know that from each matching-arc, at least one end-point of the arc must belong to H (by Lemma 18). If the number of bad M -blocks in T is more than $\frac{2k}{\log^4 k}$, then the number of vertices incident to the matching arcs is more than $2k$ which in turn implies that the number of matching arcs are more than k . So we cannot hit all the matching arcs using at most k vertices. ◀

► **Lemma 21.** *Let (T, S, k, M) be a matching instance of SUBSET-FVST. Then there exists an algorithm that in $2^{\mathcal{O}(\frac{k}{\log k})}$ time outputs a family $\mathcal{F}' = \{(M'_1, P'_1), \dots, (M'_\ell, P'_\ell)\}$ of sets with $|\mathcal{F}'| = 2^{\mathcal{O}(\frac{k}{\log k})}$ such that if there exists a S -feedback vertex set H of T of size at most k such that $H \cap M = \emptyset$ and there exists $(M', P') \in \mathcal{F}'$ satisfying:*

1. $M \cap M' = \emptyset$
2. $M' \cup P' \subseteq S$,
3. $M' \cap H = \emptyset, P' \subseteq H$,
4. In each $(M \cup M')$ -block B in $T - P'$ we have $|B \cap S| \leq 2 \log^4 k$.
5. In each $(M \cup M')$ -block B in $T - P'$ either B has no S -vertex or B is good.

Proof. Let \mathcal{B} be the set of all bad M -blocks. Let \mathcal{Q} denote the set of all S -vertices in \mathcal{B} . For each $Q \subseteq \mathcal{Q}$ such that $|Q| \leq \frac{4k}{\log^2 k}$, we output a pair of sets $(M', P') = (Q, \mathcal{Q} \setminus Q)$. The family \mathcal{F}' is the collection of all such pairs of sets.

We prove that the algorithm satisfies the stated properties. Clearly, $M' \cup P' \subseteq S$. Consider a S -feedback vertex set H of size at most k such that $H \cap M = \emptyset$. Due to the Lemma 9, in any M -block B in $T - H$, we have $|B \cap S| \leq 2 \log^2 k$. So from each bad M -block in T there are at

17:12 Subset FVS in Tournaments as Fast as Without the Subset

most $2 \log^2 k$ many non solution S -vertices. As the number of bad M -blocks in T is bounded by $\frac{2k}{\log^4 k}$ (by Observation 20), in total there are at most $\frac{2k}{\log^4 k} \times 2 \log^2 k$ many S -vertices from the union of bad M -blocks that do not belong to H . Since the algorithm loops over all choices of subsets $Q \subseteq \mathcal{Q}$, $|Q| \leq \frac{4k}{\log^2 k}$, the family \mathcal{F}' contains a pair (M', P') such that $M' \cap H = \emptyset$, $P' := \mathcal{Q} \setminus M' \subseteq H$. As in each of the M -block in T we have $|B \cap S| \leq 2 \log^4 k$ (by Lemma 12), we also have in each $M \cup M'$ -block B in $T - P'$ that $|B \cap S| \leq 2 \log^4 k$. It remains to show that in each $M \cup M'$ -block B in $T - P'$ either B has no S -triangle or B is good. Note that we only refine the partition for bad blocks. So if any $M \cup M'$ -block B is not good in $T - P'$, then it must be obtained by partitioning a bad block but in that case, we brute force over all non-solution S vertices (part of M') and delete all the solution S vertices (part of P'). So, there is no S -triangle inside (in fact, there are no S -vertices at all) if it is not good.

Now $Q \subseteq S$ and $|Q| \leq \frac{4k}{\log^2 k}$ together imply that the number of pair of sets $(Q, \mathcal{Q} \setminus Q)$ (i.e., (M', P')) is at most $(k^2)^{\frac{4k}{\log^2 k}} = 2^{2 \log k \times \frac{4k}{\log^2 k}} = 2^{\mathcal{O}(\frac{k}{\log k})}$. Hence $|\mathcal{F}'| = 2^{\mathcal{O}(\frac{k}{\log k})}$ and this can be computed in $2^{\mathcal{O}(\frac{k}{\log k})}$ time. \blacktriangleleft

Our algorithm for SUBSET-FVST applies Lemma 21 and obtains a family \mathcal{F}' . The algorithm then iterates over each pair $(M', P') \in \mathcal{F}'$ and looks for a S -feedback vertex set H of size at most k that is compatible with $(M \cup M', P')$. In particular, we delete all the vertices of P' and obtain an instance $(T - P', S \setminus P', k - |P'|, M \cup M') = (T, S, k, M)$ such that the following holds.

1. For all $v \in V(T) \setminus M$, we have that $T[M \cup \{v\}]$ does not contain a $\{v\}$ -cycle. In particular, $T[M \cup \{v\}]$ is acyclic.
2. In each M -block B in T , we have $|B \cap S| \leq 2 \log^4 k$.
3. Either each block B is good or it does not have a S -vertex.

We also need the following lemma, which will allow us to show that Lemma 21 is not applied more than once.

► Observation 22. *Let T be a tournament and $M \subseteq M^*$, then every M^* block is a refinement of M block. That is, for each M^* block B^* , there exists a block M block B such that $B^* \subseteq B$.*

This implies that if we add vertices to M to get M^* , then a good M block can be partitioned into several good blocks or if it does not have a S vertex, then none of the resulting blocks after partitioning will have a S vertex.

3.6 Branching Structure II: High Red Degree Again

In the beginning of Section 3.5, we were in the situation where red arcs only formed matching arcs in T . That is, we had (T, S, k, M) which was a matching instance. However, after application of Lemma 21, M -blocks can get partitioned and can have a set of red arcs which share a common vertex. This is possible because some local S -triangle for M -blocks in T can become a shared S -triangle for $(M \cup M')$ -blocks in $T - P'$. Thus, in this scenario, we return to the case of Section 3.4. Therefore, we can use the same branching algorithm as before in Section 3.4. However, because of Observation 22 Lemma 21 is not applied more than once.

If the previous scenario does not happen, this means that the collection of red arcs indeed form matching arcs. In that case, we proceed to Section 3.7.

3.7 Discovering Structure IV: Matching Reduction Rule

Let (T, S, k, M^*) be an instance such that the following holds.

1. For all $v \in V(T) \setminus M^*$, we have that $T[M^* \cup \{v\}]$ does not contain a $\{v\}$ -cycle. In particular, $T[M^* \cup \{v\}]$ is acyclic.
2. In each M^* -block B in T , we have $|B \cap S| \leq 2 \log^4 k$.
3. Either each block B is good or it does not have a S -vertex.
4. Red arcs form a matching in $G = (V(T), R)$.

Now observe that, any M -block of T which contains no local S -triangle still may contain end-points of too many (more than $\log^4 k$) matching arcs. In other words, T can have some bad M -blocks that have no local S -triangles. We address this issue in this section.

► **Definition 23** (inner and outer). Given a directed graph D with an ordered partition $V_1 \cup V_2 \cup \dots \cup V_t$ of $V(D)$, and an integer $i \in [t]$, we define two functions $\mathbf{inner}: \{V_1, V_2, \dots, V_t\} \rightarrow V(D)$ and $\mathbf{outer}: \{V_1, V_2, \dots, V_t\} \rightarrow V(D)$ as follows.

- $\mathbf{inner}(V_i) = \{v : (v, u) \in A(D), v \in V_i, u \in V_j, j \in [i-1]\} \cup \{v : (w, v) \in A(D), v \in V_i, w \in V_{j'}, j' \in [i+1, t]\}$
- $\mathbf{outer}(V_i) = \{u : (v, u) \in A(D), v \in V_i, u \in V_j, j \in [i-1]\} \cup \{w : (w, v) \in A(D), v \in V_i, w \in V_{j'}, j' \in [i+1, t]\}$

Notice that we are in the situation where all the red arcs are matching arcs in the ordered partition of M -blocks in T . Consider the directed graph $D = (V(T), R)$ with partitions accorded by M^* . This leads us to the following observation.

► **Observation 24.** For each M -block B in T we have $|\mathbf{inner}(B)| = |\mathbf{outer}(B)|$.

Now consider a bad block B that does not contain any local S -triangle. We will now prove that for an Yes-instance, there is always a solution of size at most k which is disjoint from $\mathbf{inner}(B)$.

► **Lemma 25.** Let H be a solution of size at most k for an instance (T, S, k, M^*) of SUBSET-FVST and let B be a bad M^* -block in T . Then $H^* := (H \setminus \mathbf{inner}(B)) \cup \mathbf{outer}(B)$ is also a solution for (T, S, k, M^*) of SUBSET-FVST with $|H^*| \leq |H|$.

Proof. By Observation 24, $|H^*| \leq |H|$. It remains to show that H^* is also a solution. In contrary, assume that H^* is not a solution. So, there exists a S -triangle denoted by Δ in $T - H^*$ and the triangle Δ must contain a vertex v from $\mathbf{inner}(B)$ where $V(\Delta)$ is disjoint from $\mathbf{outer}(B)$. Now we have following two cases.

Case 1: Δ is a shared S -triangle. Observe that this case can not happen because this triangle Δ must contain a matching-arc (red arc) whose one end point is $v \in \mathbf{inner}(B)$. Hence the other end point, say u must be in $\mathbf{outer}(B)$ which is also a vertex of the triangle Δ , which is a contradiction to the fact that $\mathbf{outer}(B) \subseteq H$.

Case 2: Δ is a local S -triangle. In this case $V(\Delta) \subseteq B$. That means the block B contains a local S -triangle which implies that B is good (by condition 5 of Lemma 21), which is a contradiction to the assumption that B is a bad M^* -block.

This concludes the proof. ◀

Now we obtain the following reduction rule whose correctness follows from Lemma 25.

► **Reduction Rule 2.** *If B is a bad M^* -block in T then we delete $\text{inner}(B) \cup \text{outer}(B)$ from T . The resultant instance is $(T - \{\text{inner}(B) \cup \text{outer}(B)\}, S \setminus \{\text{inner}(B) \cup \text{outer}(B)\}, k - |\text{outer}(B)|, M^*)$.*

After the exhaustive application of Reduction Rule 2, we have the following observation.

► **Observation 26.** *Every M^* -block in T is good.*

Currently, we are in the situation where every M^* -block in T is good, which means that the number of matching arcs incident to the vertices of every M^* -block is at most $\log^4 k$. In the following section, we will construct an undirected multigraph of bounded degree. This graph will guide us in developing a divide-and-conquer algorithm that achieves our final goal.

3.8 Balanced Edge Partitioning

In this section we deal with the case where every block is good.

Construction of a undirected multigraph. Given (T, S, k, M^*) we construct an undirected multigraph \mathcal{T} as follows:

- Each M^* -block B in T corresponds to a vertex v_B in $V(\mathcal{T})$.
- For every matching-arc (u, v) , $u \in B_i$ and $v \in B_j$ where B_i, B_j are two different M^* -blocks in T , we introduce an edge between v_{B_i} and v_{B_j} in $E(\mathcal{T})$.

At this stage, according to Observation 26, every M -block in T is good. Therefore by definition, the number of matching arcs that are incident to the vertices of every M^* -block is at most $\log^4 k$. This leads us to the next observation.

► **Observation 27.** *The maximum degree of \mathcal{T} is bounded by $\log^4 k$.*

The following fact is known regarding graph partitioning which will be useful for us.

► **Proposition 28** (Theorem 15 [19]). *Given an undirected multigraph without self-loops and isolated vertices G of maximum degree at most d and $|E(G)| = m$, there exists a partition (A, B) of $V(G)$ such that*

- $\frac{m}{4} - \frac{d}{2} \leq |E(G[A])| \leq \frac{m}{4} + \frac{d}{2}$,
- $\frac{m}{4} - \frac{d}{2} \leq |E(G[B])| \leq \frac{m}{4} + \frac{d}{2}$, and
- $\frac{m}{2} - d \leq |E(G[A, B])| \leq \frac{m}{2} + d$.

where $E(G[A, B])$ is the set of edges with one endpoint in A and other in B . Furthermore, there is a polynomial time algorithm to obtain this partition.

Recursive algorithm. We are now ready to describe our recursive algorithm to deal with good M^* -blocks. First, we construct an undirected multigraph \mathcal{T} from (T, S, k, M^*) . We then use Theorem 28 to find a balanced partition of the M^* -blocks with respect to the number of matching-arcs. We then guess which endpoints of these matching-arcs to add to H and solve the two sides independently. Our algorithm starts with an empty set $W \subseteq V(T)$, initialized as \emptyset . If \mathcal{T} is disconnected, then the algorithm solves each connected component independently and outputs W as the union of solutions returned for each component. Now we have following two cases.

Case 1. $|E(\mathcal{T})| = 0$: This case implies that there is no matching-arc in T . In that case, we solve the problem independently in each M^* -block (by brute force) and return W as the union of solutions returned for each M -block.

Case 2. $|E(\mathcal{T})| > 0$: In this case, first we run the algorithm of Proposition 28 to get a partition (A, B) of the multigraph \mathcal{T} . For a vertex set $X \subseteq V(\mathcal{T})$ and edge set $Y \subseteq E(\mathcal{T})$, the set $V_X(Y)$ denotes the set of all the vertices in T that are incident on the matching-arcs corresponding to Y and belong to M^* -blocks in X . Our algorithm loops over all subsets $C \subseteq E(A, B)$, calling itself recursively on $\mathcal{T}(V(T) \setminus (V_A(C) \cup V_B(E(A, B) \setminus C)))$ (the multigraph corresponding to the subgraph of T without $V_A(C)$ and $V_B(E(A, B) \setminus C)$) and computes $W_C := V_A(C) \cup V_B(E(A, B) \setminus C) \cup S'$ where S' is the set returned at the recursive call. Finally, the algorithm outputs the smallest set W_C over all choices of $C \subseteq E(A, B)$.

3.9 Correctness and Running Time

The correctness of the algorithm follows from each individual pieces we have proved. For the running time observe the following steps:

1. Apply kernelization algorithm in polynomial time. (Lemma 4)
2. Obtain a universal undeletable family \mathcal{M} of size $2^{\mathcal{O}(\frac{k}{\log k})}$. (Lemma 9).
3. Obtain a du-family \mathcal{F} of size $2^{\mathcal{O}(\frac{k}{\log k})}$. (Lemma 12).
4. Let $G = (V(T), R)$ be an undirected graph with the vertex set $V(T)$ and the edge set R (without orientation). Branch on vertices of red-degree at least 2 in G . This leads to (1, 2) branching and corresponds to $\mathcal{O}(1.6181^k \times k^{\mathcal{O}(1)})$.
5. Making each block B good or it does not have a S -vertex by branching in $2^{\mathcal{O}(\frac{k}{\log k})}$ ways (followed by (1, 2) branching). (Lemma 21).
6. Getting rid of blocks that do not contain any S -vertex in $k^{\mathcal{O}(1)}$ time. (Lemma 25)
7. All blocks are good. In this situation, we do divide and conquer based on Proposition 28. Now we proceed to the runtime analysis of this step of the algorithm. When $|E(\mathcal{T})| = 0$, we do brute force on the instance and that takes $k^{\mathcal{O}(\log^4 k)} = 2^{\mathcal{O}(\log^5 k)}$ time. Because when $|E(\mathcal{T})| = 0$, in each connected component the number of S -vertices gets bounded by $2 \log^4 k$ (Lemma 12), so the size of an optimal solution in each component is bounded by $2 \log^4 k$. Let $h(k, d)$ be the maximum number of leaves in the recursion tree of the algorithm when run on an input with parameter k and maximum degree d . Since in each recursive call, k decreases by at least 1, the depth of the recursion tree is at most k . In each internal node of the recursion tree, the algorithm spends $k^{\mathcal{O}(1)}$ time plus constant time for each child (which are at most $h(k, d)$) and in each leaf, it spends at most $2^{\mathcal{O}(\log^5 k)}$ time (as in leaf $|E(\mathcal{T})| = 0$). Thus, the running time of the algorithm on any input with parameters k and d is upper bounded by $h(k, d) \times 2^{\mathcal{O}(\log^5 k)} \times k^{\mathcal{O}(1)}$. To upper bound $h(k, d)$, first note that $h(a, d) + h(b, d) \leq h(a + b, d)$ because $h(a, d)$ and $h(b, d)$ represent the number of leaves of two independent sub-trees. Now for each $C \subseteq E(X, Y)$ there are no edges between A and B in $\mathcal{T}(V(T) \setminus (V_A(C) \cup V_B(E(A, B) \setminus C)))$. Hence, the algorithm effectively solves $\mathcal{T}(V(T) \setminus (V_A(C)))$ and $\mathcal{T}(V(T) \setminus (V_B(E(A, B) \setminus C)))$ independently. By Proposition 28 and since we know that we need to hit all edges in \mathcal{T} , the number of edges in $E(A, B)$ is at most $\frac{k}{4} + \frac{d}{2}$. As we have seen for each C , the algorithm calls itself twice for each C and so in total, the algorithm makes $2^{\frac{k}{2} + d + 1}$ recursive calls with parameter $\frac{k}{4} + \frac{d}{2}$ (and d does not increase). Thus $h(k, d)$ is upper-bounded by the recurrence relation $h(k, d) \leq 2^{\frac{k}{2} + d + 1} h(\frac{k}{4} + \frac{d}{2}, d)$ and $h(k, 0) = 2^{\mathcal{O}(\log^5 k)} \leq 2^{\mathcal{O}(\log^5 k)}$. This solves to $h(k, d) = 1.5874^k \cdot 2^{\mathcal{O}(d \log^5 k)}$. Hence, the running time of the algorithm is bounded by $h(k, d) \times 2^{\mathcal{O}(\log^5 k)} \times k^{\mathcal{O}(1)} = 1.5874^k \cdot 2^{\mathcal{O}(d \log^5 k)} \cdot 2^{\mathcal{O}(\log^5 k)} \cdot k^{\mathcal{O}(1)}$. As $d \leq \log^4 k$, the running time is bounded by $1.5874^k \cdot 2^{\mathcal{O}(\log^9 k)} \cdot k^{\mathcal{O}(1)}$.
8. Taking into account running time in each step we get that the algorithm runs in time $\mathcal{O}(1.6181^k + n^{\mathcal{O}(1)})$. This concludes the proof of Theorem 1.

References

- 1 Faisal N. Abu-Khzam. A kernelization algorithm for d -hitting set. *J. Comput. Syst. Sci.*, 76(7):524–531, 2010. doi:10.1016/J.JCSS.2009.09.002.
- 2 Tian Bai and Mingyu Xiao. A parameterized algorithm for subset feedback vertex set in tournaments. *Theor. Comput. Sci.*, 975:114139, 2023. doi:10.1016/J.TCS.2023.114139.
- 3 Stéphane Bessy, Marin Bougeret, Dimitrios M Thilikos, and Sebastian Wiederrecht. Kernelization for graph packing problems via rainbow matching. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3654–3663. SIAM, 2023. doi:10.1137/1.9781611977554.CH139.
- 4 Jianer Chen, Yang Liu, Songjian Lu, Barry O’Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5):21:1–21:19, 2008. doi:10.1145/1411509.1411511.
- 5 Rajesh Hemant Chitnis, Marek Cygan, Mohammad Taghi Hajiaghayi, and Dániel Marx. Directed subset feedback vertex set is fixed-parameter tractable. *ACM Trans. Algorithms*, 11(4):28:1–28:28, 2015. doi:10.1145/2700209.
- 6 Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. Subset feedback vertex set is fixed-parameter tractable. *SIAM J. Discret. Math.*, 27(1):290–309, 2013. doi:10.1137/110843071.
- 7 Michael Dom, Jiong Guo, Falk Hüffner, Rolf Niedermeier, and Anke Truß. Fixed-parameter tractability results for feedback set problems in tournaments. *J. Discrete Algorithms*, 8(1):76–86, 2010. doi:10.1016/J.JDA.2009.08.001.
- 8 Guy Even, Joseph Naor, and Leonid Zosin. An 8-approximation algorithm for the subset feedback vertex set problem. *SIAM J. Comput.*, 30(4):1231–1252, 2000. doi:10.1137/S0097539798340047.
- 9 Fedor V. Fomin, Tien-Nam Le, Daniel Lokshtanov, Saket Saurabh, Stéphan Thomassé, and Meirav Zehavi. Subquadratic kernels for implicit 3-hitting set and 3-set packing problems. *ACM Trans. Algorithms*, 15(1):13:1–13:44, 2019. doi:10.1145/3293466.
- 10 Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: theory of parameterized preprocessing*. Cambridge University Press, 2019.
- 11 Jiong Guo and Rolf Niedermeier. Invitation to data reduction and problem kernelization. *SIGACT News*, 38(1):31–45, 2007. doi:10.1145/1233481.1233493.
- 12 Sushmita Gupta, Sounak Modak, Saket Saurabh, and Sanjay Seetharaman. Quick-sort style approximation algorithms for generalizations of feedback vertex set in tournaments. In José A. Soto and Andreas Wiese, editors, *LATIN 2024: Theoretical Informatics - 16th Latin American Symposium, Puerto Varas, Chile, March 18-22, 2024, Proceedings, Part I*, volume 14578 of *Lecture Notes in Computer Science*, pages 225–240. Springer, 2024. doi:10.1007/978-3-031-55598-5_15.
- 13 Eva-Maria C. Hols and Stefan Kratsch. A randomized polynomial kernel for subset feedback vertex set. *Theory Comput. Syst.*, 62(1):63–92, 2018. doi:10.1007/S00224-017-9805-6.
- 14 Yoichi Iwata and Yusuke Kobayashi. Improved analysis of highest-degree branching for feedback vertex set. *Algorithmica*, 83(8):2503–2520, 2021. doi:10.1007/S00453-021-00815-W.
- 15 Yoichi Iwata, Magnus Wahlström, and Yuichi Yoshida. Half-integrality, lp-branching, and FPT algorithms. *SIAM J. Comput.*, 45(4):1377–1411, 2016. doi:10.1137/140962838.
- 16 Yoichi Iwata, Yutaro Yamaguchi, and Yuichi Yoshida. 0/1/all csps, half-integral a-path packing, and linear-time FPT algorithms. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 462–473. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00051.
- 17 Satyabrata Jana, Daniel Lokshtanov, Soumen Mandal, Ashutosh Rai, and Saket Saurabh. Parameterized approximation scheme for feedback vertex set. In Jérôme Leroux, Sylvain Lombardy, and David Peleg, editors, *48th International Symposium on Mathematical Foundations of Computer Science, MFCS 2023, August 28 to September 1, 2023, Bordeaux, France*, volume 272 of *LIPICs*, pages 56:1–56:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.MFCS.2023.56.

- 18 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.
- 19 Mithilesh Kumar and Daniel Lokshтанov. Faster exact and parameterized algorithm for feedback vertex set in tournaments. In Nicolas Ollinger and Heribert Vollmer, editors, *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, volume 47 of *LIPICs*, pages 49:1–49:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.STACS.2016.49.
- 20 Jason Li and Jesper Nederlof. Detecting feedback vertex sets of size k in $O^*(2.7k)$ time. *ACM Trans. Algorithms*, 18(4):34:1–34:26, 2022. doi:10.1145/3504027.
- 21 Rolf Niedermeier and Peter Rossmanith. An efficient fixed-parameter algorithm for 3-hitting set. *Journal of Discrete Algorithms*, 1(1):89–102, 2003. doi:10.1016/S1570-8667(03)00009-1.
- 22 Geevarghese Philip, Varun Rajan, Saket Saurabh, and Prafullkumar Tale. Subset feedback vertex set in chordal and split graphs. *Algorithmica*, 81(9):3586–3629, 2019. doi:10.1007/S00453-019-00590-9.
- 23 Magnus Wahlström. *Algorithms, measures and upper bounds for satisfiability and related problems*. PhD thesis, Linköping University, Sweden, 2007. URL: <https://nbn-resolving.org/urn:nbn:se:liu:diva-8714>.

Parameterised Distance to Local Irregularity

Foivos Fioravantes 

Department of Theoretical Computer Science, FIT,
Czech Technical University in Prague, Czech Republic

Nikolaos Melissinos 

Department of Theoretical Computer Science, FIT,
Czech Technical University in Prague, Czech Republic

Theofilos Triommatis 

School of Electrical Engineering, Electronics and Computer Science University of Liverpool, UK

Abstract

A graph G is *locally irregular* if no two of its adjacent vertices have the same degree. The authors of [Fioravantes et al. Complexity of finding maximum locally irregular induced subgraph. *SWAT*, 2022] introduced and provided some initial algorithmic results on the problem of finding a locally irregular induced subgraph of a given graph G of maximum order, or, equivalently, computing a subset S of $V(G)$ of minimum order, whose deletion from G results in a locally irregular graph; S is called an *optimal vertex-irregulator* of G . In this work we provide an in-depth analysis of the parameterised complexity of computing an optimal vertex-irregulator of a given graph G . Moreover, we introduce and study a variation of this problem, where S is a subset of the edges of G ; in this case, S is denoted as an *optimal edge-irregulator* of G . We prove that computing an optimal vertex-irregulator of a graph G is in FPT when parameterised by various structural parameters of G , while it is W[1]-hard when parameterised by the feedback vertex set number or the treedepth of G . Moreover, computing an optimal edge-irregulator of a graph G is in FPT when parameterised by the vertex integrity of G , while it is \mathcal{NP} -hard even if G is a planar bipartite graph of maximum degree 6, and W[1]-hard when parameterised by the size of the solution, the feedback vertex set or the treedepth of G . Our results paint a comprehensive picture of the tractability of both problems studied here.

2012 ACM Subject Classification Mathematics of computing \rightarrow Graph algorithms; Theory of computation \rightarrow Parameterized complexity and exact algorithms

Keywords and phrases Locally irregular, largest induced subgraph, FPT, W-hardness

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.18

Related Version *Full Version:* <https://arxiv.org/abs/2307.04583>

Funding This work was co-funded by the European Union under the project Robotics and advanced industrial production (reg. no. CZ.02.01.01/00/22_008/0004590). Foivos Fioravantes and Nikolaos Melissinos are also supported by the CTU Global postdoc fellowship program.

Acknowledgements The authors would like to thank Dániel Marx for his important contribution towards rendering the proof of Theorem 7 more elegant.

1 Introduction

A fundamental problem in graph theory is “given a graph G , find an induced subgraph H of G , of maximum order, that belongs in the family of graphs verifying a property Π ”, in which case we say that $H \in \Pi$:

LARGEST INDUCED SUBGRAPH WITH PROPERTY Π (ISP- Π)[19]

Input: A graph $G = (V, E)$, an integer k , a property Π .

Task: Does there exist a set $S \subseteq V$ such that $|S| \leq k$ and $G - S \in \Pi$?



© Foivos Fioravantes, Nikolaos Melissinos, and Theofilos Triommatis;
licensed under Creative Commons License CC-BY 4.0

19th International Symposium on Parameterized and Exact Computation (IPEC 2024).

Editors: Édouard Bonnet and Paweł Rzażewski; Article No. 18; pp. 18:1–18:15

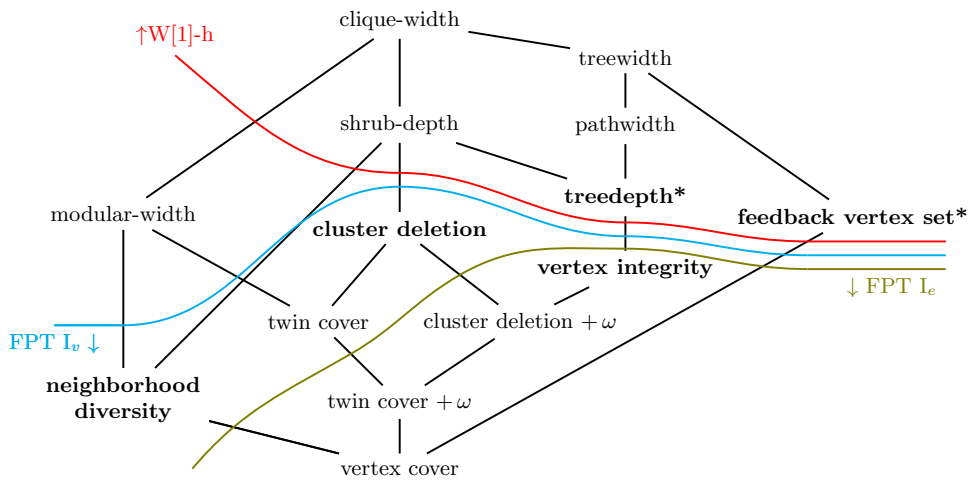
Leibniz International Proceedings in Informatics



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

There is a plethora of classical problems that fall under this general setting. Consider, for example, the VERTEX COVER and the FEEDBACK VERTEX SET, where Π is the property “the graph is an independent set” and “the graph is a forest”, respectively.

In this paper we study the ISP- Π problem where Π is the property “the graph is locally irregular”, recently introduced in [16]. A graph $G = (V, E)$ is called *locally irregular* if no two adjacent vertices in V have the same degree. We extend the work of [16], by more thoroughly investigating the parameterised behaviour of the problem. In addition, we take the first step towards the problem of finding large locally irregular (not necessarily induced) subgraphs of a given graph G . In particular, we introduce the problem where the goal is to find a subset of edges of G of minimum order, whose removal renders the graph locally irregular. Our results allow us to paint a rather clear picture concerning the tractability of both problems studied here in relation to various standard graph-structural parameters (see Figure 1 for an overview of our results).



■ **Figure 1** Overview of our results. A parameter A appearing linked to a parameter B with A being below B is to be understood as “there is a function f such that $f(A) \geq f(B)$ ”. The bold font is used to indicate the parameters that we consider in this work. The asterisks are used to indicate that the corresponding result follows from observations based on the work in [16]. In light blue (olive resp.) we exhibit the FPT results we provide for finding an optimal vertex (edge resp.) irregularator. In red we exhibit the $W[1]$ -hardness results we provide for both problems. The clique number of the graph is denoted by ω .

ISP- Π and heredity. The ISP- Π problem has been extensively studied for *hereditary* properties. That is, a property Π is hereditary if, for any graph G verifying it, any induced subgraph of G also verifies that property. The properties “the graph is an independent set” or “the graph is a forest” are, for example, hereditary. It was already shown in [29] that ISP- Π is an \mathcal{NP} -hard problem for any non-trivial hereditary property. On the positive side, the ISP- Π problem always admits an FPT algorithm, when parameterised by the size of the solution, if Π is a hereditary property [8, 25]. This is an important result, as it allows us to conceive efficient algorithms to solve computationally hard problems, as long as we restrict ourselves to graphs verifying such properties.

It is also worth mentioning the work in [17], which provides a framework that yields exact algorithms that are significantly faster than brute-force to solve a more general version of the ISP- Π problem: given a universe, find a subset of maximum cardinality which verifies some

hereditary property. On a high level, the algorithm proposed in [17] builds the solution which is a subset H of maximum cardinality with the wanted property, by continuously extending a partial solution $X \subseteq H$. Note that this approach only works if Π is indeed a hereditary property. More recently, this approach was generalised by the authors of [14], who provide a framework that yields exponential-time approximation algorithms.

However, not all interesting properties are hereditary. E.g., “all vertices of the induced subgraph have odd degree”, and “the induced subgraph is d -regular”, where d is an integer given in the input (recall that a graph is d -regular if all of its vertices have the same degree d), are non-hereditary properties. The authors of [5] studied the ISP-II problem for the former property, showing that it is an \mathcal{NP} -hard problem, and providing an FPT algorithm that solves it when parameterised by the rank-width. Also, the authors of [1, 3, 31] studied the ISP-II problem for the latter property. It is shown in [3] that finding an induced subgraph of maximum order that is d -regular is \mathcal{NP} -hard to approximate, even on bipartite or planar graphs. The authors of [3] also provide a linear-time algorithm to solve this problem for graphs with bounded treewidth. Lastly, it is also worth mentioning [7], where the authors consider the non-hereditary property “the induced subgraph is k -anonymous”, where a graph G is k -anonymous if for each vertex of G there are at least $k - 1$ other vertices of the same degree.

An important observation is that, in the case of non-hereditary properties, the ISP-II problem does not necessarily admit an FPT algorithm parameterised by the size of the solution. Indeed, the authors of [31] proved that when considering Π as “the induced subgraph is regular”, the ISP-II problem is $W[1]$ -hard when parameterised by the size of the solution. This indicates the importance of considering graph-structural parameters for conceiving efficient algorithms for such problems. This is exactly the approach followed in [18, 27], where the authors consider a generalisation of VERTEX COVER, the ISP-II problem where Π is “the graph has maximum degree k ”, for an integer k given in the input.

Distance from local irregularity. In some sense, the property that interests us lies on the opposite side of the one studied in [1, 3, 31]. Recall that a graph G is locally irregular if no two of its adjacent vertices have the same degrees. This notion was formally introduced in [4], where the authors take some steps towards proving the so-called 1-2-3 Conjecture proposed in [23] and recently proven in [24]. Roughly, this conjecture is about functions assigning weights from $[k] = \{1, \dots, k\}$ to the edges of a graph, called proper k -labellings, so that all adjacent vertices have different weighted degrees; the conjecture states that for any non-trivial graph, this is always achievable for $k \leq 3$.

The authors of [16] introduced the problem of finding a locally irregular induced subgraph of a given graph G of maximum order (a non-hereditary property). Equivalently, find a set of *vertices* of minimum cardinality whose deletion renders the given graph locally irregular; such sets are named *optimal vertex-irregularators*. The main focus of [16] was to study the complexity of computing optimal vertex-irregularators. It was shown that this problem is \mathcal{NP} -hard even for subcubic planar bipartite graphs, $W[2]$ -hard parameterised by the size of the solution and $W[1]$ -hard parameterised by the treewidth of the input graph. Moreover, for any constant $\varepsilon < 1$, there cannot be a polynomial-time $\mathcal{O}(n^{1-\varepsilon})$ -approximation algorithm (unless $\mathcal{P}=\mathcal{NP}$). On the positive side, there are two FPT algorithms that solve this problem, parameterised by the maximum degree of the input graph plus either the size of the solution or the treewidth of the input graph. Note that the notion of vertex-irregularators proved to be fruitful in the context of proper labellings. Indeed, the authors of [6] observed a connection between finding large locally irregular induced subgraphs and constructing proper k -labellings that also maximise the use of weight 1 on the edges of the given graph.

Apart from improving the results of [16], in this paper we also introduce the novel problem of computing a subset of a graph's *edges*, of minimum order, whose deletion renders the graph locally irregular; such sets are named *optimal edge-irregularators*. This problem is introduced as a first step towards understanding the problem of finding large locally irregular (not necessarily induced) subgraphs of a given graph. Problems concerned with finding maximum subgraphs verifying a specific property have also been extensively studied (*e.g.*, [9, 10, 2]). One might expect that finding edge-irregularators could be easier than finding vertex-irregularators as it is often the case with graph theoretical problems concerned with subsets of edges, whose versions considering subsets of vertices are intractable (recall, *e.g.*, EDGE COVER, FEEDBACK EDGE SET and even MIN WEIGHTED LOWER-UPPER-COVER [33]). As it turns out, however, finding small edge-irregularators is also a computationally hard problem.

Our contribution. In this paper we study the complexity of computing optimal vertex and edge-irregularators. We identify the parameters for which the tractability of the former problem changes, considering a multitude of standard graph-structural parameters. We also take steps towards the same goal for the latter problem. In Section 2 we introduce the needed notation and provide some first results. In particular, we observe that computing optimal vertex-irregularators is W[1]-hard when parameterised by the treedepth or the feedback vertex set of the given graph. Section 3 provides FPT algorithms for the problem of finding optimal vertex-irregularators. The considered parameters are the neighborhood diversity, the vertex integrity, or the clustered deletion number of the input graph. In Section 4, we focus on the problem of finding optimal edge-irregularators. First, we prove that this problem is \mathcal{NP} -hard, even when restricted to planar bipartite graphs of maximum degree 6. We also show that the problem is W[1]-hard parameterised by the size of the solution or the feedback vertex set of the input graph. Lastly, we modify the FPT algorithm for computing an optimal vertex-irregularator parameterised by the vertex integrity in order to provide an FPT algorithm that solves the edge version of the problem (once more parameterised by the vertex integrity). We close the paper in Section 5, where we propose some directions for further research.

2 Preliminaries

We follow standard graph theory notations [12].

Let $G = (V, E)$ be a graph and $G' = (V', E')$ be a subgraph of G (*i.e.*, created by deleting vertices and/or edges of G). Recall first that the subgraph G' is *induced* if it can be created only by deleting vertices of G ; in this case we denote G' by $G[V']$. That is, for each edge $uv \in E$, if $u, v \in V'$, then $uv \in E'$. For any vertex $v \in V$, let $N_G(v) = \{u \in V : uv \in E\}$ denote the *neighbourhood* of v in G and $d_G(v) = |N_G(v)|$ denote the *degree* of v in G . Note that, whenever the graph G is clear from the context, we will omit the subscript and simply write $N(v)$ and $d(v)$. Also, for $S \subseteq E$, we denote by $G - S$ the graph $G' = (V, E \setminus S)$. That is, G' is the graph resulting from the deletion of the edges of S from the graph G .

Let $G = (V, E)$ be a graph. We say that G is *locally irregular* if, for every edge $uv \in E$, we have $d(u) \neq d(v)$. Now, let $S \subseteq V$ be such that $G[V \setminus S]$ is a locally irregular graph; any set S that has this property is denoted as a *vertex-irregularator* of G . Moreover, let $I_v(G)$ be the minimum order that any vertex-irregularator of G can have. We will say that S is an *optimal* vertex-irregularator of G if S is a vertex-irregularator of G and $|S| = I_v(G)$. Similarly, we define an *edge-irregularator* of G to be any set $S \subseteq E$ such that $G - S$ is locally irregular. Moreover, let $I_e(G)$ be the minimum order that any edge-irregularator of G can have. We will say that S is an *optimal* edge-irregularator of G if S is an edge-irregularator of G and $|S| = I_e(G)$.

We begin with some simple observations that hold for any graph $G = (V, E)$.

► **Observation 1.** *If G contains two vertices u, v such that $uv \in E$ and $d(u) = d(v)$, then any edge-irregulator of G contains at least one edge incident to u or v . Also, any vertex-irregulator of G contains at least one vertex in $N(u) \cup N(v)$.*

► **Observation 2.** *If G contains two vertices $u, v \in V$ that are twins, i.e., $N(u) \setminus \{v\} = N(v) \setminus \{u\}$, such that $uv \in E$, then any vertex-irregulator of G contains at least one vertex in $\{u, v\}$.*

The importance of upcoming Lemma 3 lies in the fact that we can repeatedly apply it, reducing the size of the graph on which we are searching for a vertex-irregulator. This is a core argument behind the algorithms of Theorems 7 and 11.

► **Lemma 3.** *Let $G = (V, E)$ be a graph and $u, v \in V$ be a pair of adjacent twins. Let $G' = (V', E')$ be the graph resulting from the deletion of either u or v from G . Then, $I_v(G) = I_v(G') + 1$.*

Proof. Assume w.l.o.g. that $u \notin V'$. We first prove that $I_v(G) \leq I_v(G') + 1$. Indeed, assume that $I_v(G) > I_v(G') + 1$ and let S' be an optimal vertex-irregulator of G' . Next, consider the graph $\tilde{G} = G[V \setminus (S' \cup \{u\})]$. From the construction of G' , it follows that $\tilde{G} = G'[V' \setminus S']$. Since S' is a vertex-irregulator of G' , we obtain that \tilde{G} is locally irregular. In other words, the set $S' \cup \{u\}$ is a vertex-irregulator of G and $|S' \cup \{u\}| = I_v(G') + 1$, a contradiction.

Next, assume that $I_v(G) < I_v(G') + 1$ and let S be an optimal vertex-irregulator of G . It follows from Observation 2 that $|\{u, v\} \cap S| \geq 1$. Assume w.l.o.g. that $u \in S$. Thus, and by the construction of G' , we have that $G'[V' \setminus (S \setminus \{u\})] = G[V \setminus S]$ and the set $S \setminus \{u\}$ is a vertex-irregulator of G' . In other words, $I_v(G') \leq |S| - 1 = I_v(G) - 1$, a contradiction. ◀

We close this section with some observations on the proof that computing $I_v(G)$ is $W[1]$ -hard parameterised by the treewidth of G , initially presented in [16], which allows us to show that this result holds even if we consider more “restricted” parameters, such as the treedepth or the feedback vertex set number (i.e., size of a minimum feedback vertex set) of the input graph. Recall that the *treedepth* of a graph $G = (V, E)$ can be defined recursively: if $|V| = 1$, then G has treedepth 1. Then, G has treedepth k if there exists a vertex $v \in V$ such that every connected component of $G[V \setminus \{v\}]$ has treedepth at most $k - 1$. Given a graph G and a tree T rooted at a vertex u , by *attaching* T on a vertex v of G , we mean the operation of adding T to G and identifying u with v , i.e., $V(T) \cap V(G) = \{u\} = \{v\}$.

► **Observation 4.** *Let G be a graph with vertex cover number (i.e., size of a minimum vertex cover) k_1 and T be a rooted tree of depth k_2 . Let G' be the graph after attaching an arbitrary number of copies of T directly on vertices of G . Then G' has treedepth $\mathcal{O}(k_1 + k_2)$ and feedback vertex set number $\mathcal{O}(k_1)$.*

The reduction presented in [16, Theorem 16] starts with a graph G which is part of an instance of the LIST COLOURING problem, and constructs a graph G' by attaching some trees of depth at most 3 on each vertex of G . The LIST COLOURING problem was shown to be $W[1]$ -hard in [15] when parameterised by the vertex cover number of the input graph. Thus, by Observation 4, we obtain the following:

► **Corollary 5.** *Given a graph G , it is $W[1]$ -hard to compute $I_v(G)$ parameterised by either the treedepth or the feedback vertex set number of G .*

3 FPT algorithms for vertex-irregularity

In this section we present two FPT algorithms that compute an optimal vertex-irregularity of a given graph G , when parameterised by the neighbourhood diversity or the vertex integrity of G . The latter algorithm is then used to show that this problem is in FPT also when parameterised by the cluster deletion number of G . We begin by recalling the needed definitions.

The *twin equivalence* of G is the relation on the vertices of V where two vertices belong to the same equivalence class if and only if they are twins.

► **Definition 6** ([26]). *A graph G has neighbourhood diversity k (denoted as $nd(G) = k$) if its twin equivalence has k classes.*

Let $G = (V, E)$ be a graph with $nd(G) = k$ and let V_1, \dots, V_k be the partition of V defined by the twin equivalence of G . Observe that for any $i \in [k]$, we have that $G[V_i]$ is either an independent set or a clique.

► **Theorem 7.** *Given a graph $G = (V, E)$ such that $nd(G) = k$, there exists an algorithm that computes $I_v(G)$ in FPT-time parameterised by k .*

Proof. Let V_1, \dots, V_k be the partition of V defined by the twin equivalence of G . Note that this partition can be computed in linear time [26]. We begin by constructing an induced subgraph $G' = (V', E')$ of G by applying the following procedure: for each $i \in [k]$, if $G[V_i]$ is a clique on at least two vertices, delete all the vertices of V_i except one; let D be the set of vertices that were deleted and $d = |D|$. This procedure terminates after k iterations and, thus, runs in polynomial time. Moreover, it follows from Lemma 3 that $I_v(G) = I_v(G') + d$. Thus, it suffices to solve the problem on G' . For every $i \in [k]$, let $V'_i = V_i \cap V'$.

Observe that for every locally irregular graph H , there exists a prime number p such that $d_H(u) - d_H(v) \not\equiv 0 \pmod p$ for every $uv \in E(H)$. In our case, since for every $uv \in E'$ we have that $u \in V'_i$ and $v \in V'_j$ for $i < j \leq [k]$, it follows that there can be at most $\binom{k}{2}$ possible differences modulo p between the degrees of adjacent vertices in G^* , where $G^* = (V^*, E^*)$ is a locally irregular induced subgraph of G' .

We claim that $p \leq (k^2 \log n + 1)(\log(k^2 \log n + 1) + \log \log(k^2 \log n + 1) - \frac{1}{2})$. Indeed, since each one of the differences we considered in the previous paragraph is at most n , each one of them has at most $\log n$ prime divisors. Thus, p is at most the $k^2 \log n + 1^{\text{th}}$ prime number. This, in conjunction with the classical results from [32] gives us the claimed inequality.

For every $i \in [k]$, let $V_i^* = V'_i \cap V^*$. For every such prime p , we consider all the possible cases for $p_i = |V_i^*| \pmod p$, for every $i \in [k]$; there are at most p^k such instances. Let us consider any such instance such that $d_{G^*}(u) - d_{G^*}(v) \not\equiv 0 \pmod p$ for every $uv \in E^*$. Checking this inequality is straightforward from the p_i s. We store the maximum orders of the V_i^* s such that $|V_i^*| \pmod p = p_i$ for every $i \in [k]$. Having repeated this procedure for all such instances, we are certain to have computed a locally irregular induced subgraph of G' of maximum order. In total, this procedure takes time $p^{k+1}n^{\mathcal{O}(1)}$ which is in FPT due to the upper bound on p by [32] and since $\log^k n \leq f(k)n$, for some computable function f [22]. ◀

We now present an FPT algorithm to compute an optimal vertex-irregularity of an input graph G when parameterised by the vertex integrity of G , which can be computed in FPT-time [13].

► **Definition 8.** *A graph $G = (V, E)$ has vertex integrity k if there exists a set $U \subseteq V$ such that $|U| = k' \leq k$ and all connected components of $G[V \setminus U]$ are of order at most $k - k'$.*

► **Theorem 9.** *Given a graph $G = (V, E)$ with vertex integrity k , there exists an algorithm that computes $I_v(G)$ in FPT-time parameterised by k .*

Proof. Let $U \subseteq V$ be such that $|U| = k' \leq k$ and C_1, \dots, C_m be the vertex sets of the connected components of $G[V \setminus U]$ such that $|C_j| \leq k - k', j \in [m]$. Assume that we know the intersection of an optimal vertex-irregulator S of G and the set U , and let $S' = S \cap U$ and $U' = U \setminus S$ (there are at most $2^{|U|} \leq 2^k$ possible intersections S' of U and S). Notice that the graph $G[V \setminus S']$ has an optimal vertex-irregulator that contains only vertices from $\bigcup_{i \in [m]} C_i$. Indeed, assuming otherwise contradicts that S' is the intersection of an optimal vertex-irregulator and U . Thus, in order to find an optimal vertex-irregulator S of G , it suffices to compute $S^* \subseteq \bigcup_{i \in [m]} C_i$, which is an optimal vertex-irregulator of $G[V \setminus S']$, for every set $S' \subseteq U$. Then, we return the set $S^* \cup S'$ of minimum order. We compute S^* through an ILP with bounded number of variables. To do so, we define types and sub-types of graphs $G[U' \cup C_j], j \in [m]$.

Informally, the main idea is to categorise the graphs $G[U' \cup C_j], j \in [m]$, into types based on their structure (formally defined later), whose number is bounded by some function of k . Each type i is associated with a number no_i that represents the number of the subgraphs $G[U' \cup C_j], j \in [m]$, that belong in that type. Then, for each type i , we will define sub-types based on the induced subgraphs $G[(U' \cup C_j) \setminus S_q]$, for $S_q \subseteq C_j$. We also define a variable $no_{i,q}$ that is the number of the subgraphs $G[U' \cup C_j], j \in [m]$, that are of type i and of sub-type q in $G[V \setminus S]$. Note that knowing the structure of these types and sub-types, together with $no_{i,q}$, is enough to compute the order of S^* . Finally, for any $j \in [m]$, the graph $G[U' \cup C_j]$ is of order at most k . Thus, the number of types, sub-types and their corresponding variables, is bounded by a function of k . We will present an ILP formulation whose objective is to minimise the order of S^* .

We begin by defining the types. Two graphs $G[U' \cup C_i]$ and $G[U' \cup C_j], i, j \in [m]$, are of the same type if there exists a bijection¹ $f : C_i \cup U' \rightarrow C_j \cup U'$ such that $f(u) = u$ for all $u \in U'$ and $N_{G[U' \cup C_i]}(u) = \{f^{-1}(v) \mid v \in N_{G[U' \cup C_j]}(f(u))\}$ for all $u \in C_i$. Note that if such a function exists, then $G[U' \cup C_i]$ is isomorphic to $G[U' \cup C_j]$.

Let p be the number of different types. Notice that p is bounded by a function of k as any graph $G[U' \cup C_i]$ has order at most k . Also, we can decide if two graphs $G[U' \cup C_i]$ and $G[U' \cup C_j], i, j \in [m]$, are of the same type in FPT-time parameterised by k . For each type $i \in [p]$, set no_i to be the number of graphs $G[U' \cup C_j], j \in [m]$, of type i . Furthermore, for each type $i \in [p]$ we select a $C_j, j \in [m]$, such that $G[U' \cup C_j]$ is of type i , to represent that type; we will denote this set of vertices by R_i .

We are now ready to define the sub-types. Let $i \in [p]$ be a type represented by R_i and $S_1^i, \dots, S_{2^{|R_i|}}^i$ be an enumeration of the subsets of R_i . For any $q \in [2^{|R_i|}]$, we define a sub-type (i, q) which represents the induced subgraph $G[(U' \cup R_i) \setminus S_q^i]$. Let $no_{i,q}$ be the variable corresponding to the number of graphs represented by $G[U' \cup R_i], i \in [p]$, that is of type (i, q) in $G[V \setminus S^*]$, for a vertex-irregulator S^* such that $S^* \cap R_i = S_q^i$.

Notice that, given a vertex-irregulator $S^* \subseteq \bigcup_{j \in [m]} C_j$ of $G[V \setminus S']$, there exists a sub-type $(i, q), i \in [p], q \in [2^{|R_i|}]$, for each $j \in [m]$, such that the graph $G[(U' \cup C_j) \setminus S^*]$ is of sub-type (i, q) . Also, assuming that we know the order of $|S_q^i|$ and the number $no_{i,q}$ for all $i \in [p], q \in [2^{|R_i|}]$, then $|S^*| = \sum_{i \in [p]} \sum_{q \in [2^{|R_i|}]} no_{i,q} |S_q^i|$.

¹ Recall that a function $f : A \rightarrow B$ is a *bijection* if, for every $a_1, a_2 \in A$ with $a_1 \neq a_2$, we have that $f(a_1) \neq f(a_2)$ and for every $b \in B$, there exists an $a \in A$ such that $f(a) = b$. Recall also that the *inverse* function of f , denoted as f^{-1} , exists if and only if f is a bijection, and is such that $f^{-1} : B \rightarrow A$ and for each $b \in B$ we have that $f^{-1}(b) = a$, where $f(a) = b$.

18:8 Parameterised Distance to Local Irregularity

Before giving the ILP formulation whose goal is to find a vertex-irregularator S^* while minimising the above sum, we guess the (i, q) such that $no_{i,q} \neq 0$. Let S_2 be the set of pairs (i, q) , $i \in [p]$ and $q \in [2^{|R_i|}]$, such that there are two vertices $u, v \in R_i \setminus S_q^i$ where $uv \in E(G[(U' \cup C_i) \setminus S_q^i])$ and $d_{G[(U' \cup R_i) \setminus S_q^i]}(u) = d_{G[(U' \cup R_i) \setminus S_q^i]}(v)$. For every $(i, q) \in S_2$, we have that $no_{i,q} = 0$. Indeed, assuming otherwise contradicts the fact that S^* is a vertex-irregularator. We guess $S_1 \subseteq \{(i, q) \mid i \in [p], q \in [2^{|R_i|}]\} \setminus S_2$ such that $no_{i,q} \neq 0$ for all $(i, q) \in S_1$. Observe that the number of different sets that are candidates for S_1 is bounded by some function of k .

Constants		
no_i	$i \in [p]$	number of components of type i
$e_{uv} \in \{0, 1\}$	$u, v \in U'$	set to 1 iff $uv \in E(G[U'])$
$e_{u,v}^{i,q} \in \{0, 1\}$	$i \in [p], q \in [2^{ R_i }], u \in U'$ and $v \in R_i \setminus S_q^i$	set to 1 iff $uv \in E(G[(U' \cup R_i) \setminus S_q^i])$
$b_u^{i,q} \in [n]$	$i \in [p], q \in [2^{ R_i }]$ and $u \in U'$	set to $d_{G[(\{u\} \cup R_i) \setminus S_q^i]}(u)$
$d_u^{i,q} \in [n]$	$i \in [p], q \in [2^{ R_i }]$ and $u \in R_i \setminus S_q^i$	set to $d_{G[(U' \cup R_i) \setminus S_q^i]}(u)$
Variables		
$no_{i,q}$	$i \in [p], q \in [2^{ R_i }]$	number of graphs of types (i, q)
Objective		
$\min \sum_{i \in [p]} \sum_{q \in [2^{ R_i }]} no_{i,q} S_q^i $		(3.1)
Constraints		
$no_{i,q} = 0$		iff $(i, q) \notin S_1$ (3.2)
$\sum_{q \in [2^{ R_i }]} no_{i,q} = no_i$		$\forall i \in [p]$ (3.3)
$\sum_{w \in U'} e_{wv} + \sum_{i \in [p]} no_{i,q} b_v^{i,q} \neq \sum_{w \in U'} e_{wu} + \sum_{i \in [p]} no_{i,q} b_u^{i,q}$		$\forall u, v \in U', e_{uv} = 1$ (3.4)
$d_v^{i,q} \neq \sum_{w \in U'} e_{wu} + \sum_{i \in [p]} no_{i,q} b_u^{i,q}$		$\forall e_{u,v}^{i,q} = 1$ and $(i, q) \in S_1$ (3.5)

Assume that we have found the values $no_{i,q}$ for (i, q) , $i \in [p]$, $q \in [2^{|R_i|}]$. We construct an optimal vertex-irregularator of $G[V \setminus S']$ as follows. Start with an empty set S^* . For each $i \in [p]$ take all components C_j of type i . Partition them into $2^{|R_i|}$ sets C_q^i such that any set $q \in [2^{|R_i|}]$ contains exactly $no_{i,q}$ of these components. For any component $C \in C_q^i$, select all vertices represented by the set S_q^i (as it was defined before) and add them to S^* . The final S^* is an optimal vertex-irregularator for $G[V \setminus S']$.

Let $S = S' \cup S^*$. We show that S is a vertex-irregularator of G . To do so, it suffices to verify that in the graph $G[V \setminus S]$ there are no two adjacent vertices with the same degree. Let u, v be a pair of adjacent vertices in a component represented by $R_i \setminus S$, which is of type (i, q) . If $d_{G[V \setminus S]}(u) = d_{G[V \setminus S]}(v)$, then $(i, q) \in S_2$. Therefore, $no_{i,q} = 0$ and we do not have such a component in $G[V \setminus S]$. Thus, it suffices to focus on adjacent vertices such that at

least one of them is in U' . Notice that, in $G[V \setminus S]$, the degree of vertex $u \in U'$ is equal to $\sum_{w \in U'} e_{uw} + \sum_{i \in [p]} n o_{i,q} b_v^{i,q}$. In other words, no two adjacent vertices in U' have the same degree due to the constraint 3.4. Lastly, the constraint 3.5 guarantees that no vertex in U' is adjacent to a vertex in $C_i \setminus S$ (for some $i \in [p]$) such that both of them have the same degree in $G[V \setminus S]$. Moreover, both S' and S^* are constructed to be minimum such sets. Thus, S is an optimal vertex-irregulator of G . Finally, since the number of variables in the model is bounded by a function of k , we can obtain S^* in FPT time parameterised by k (by running for example the Lenstra algorithm [28]). ◀

The previous algorithm can be used to find an optimal vertex-irregulator of a graph G in FPT-time when parameterised by the cluster deletion number of G . Note that the cluster deletion number of a graph can be computed in FPT-time parameterised by k [21].

► **Definition 10** ([21]). *A graph $G = (V, E)$ has cluster deletion number k if there exists a set $S \subseteq V$ such that all the connected components of $G[V \setminus S]$ are cliques, and S is of order at most k .*

► **Theorem 11.** *Given a graph $G = (V, E)$ with cluster deletion number k , there exists an algorithm that computes $I_v(G)$ in FPT-time parameterised by k .*

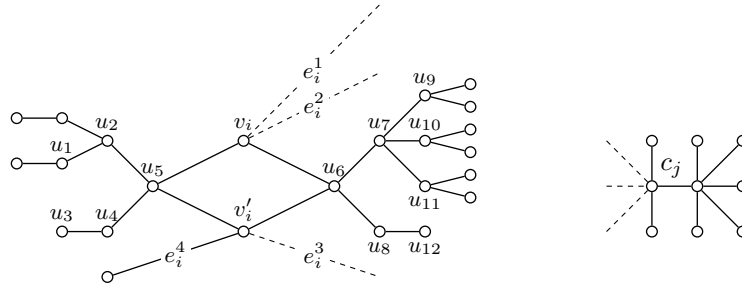
Proof. Let S be such that $|S| = k$ and $G[V \setminus S]$ is a disjoint union of cliques C_1, \dots, C_m for $m \geq 1$. Our goal is to reduce the size of these cliques so that each one of them has order at most 2^k . We achieve this through the following procedure. Let $i \in [m]$ be such that the clique $C_i = (V_{C_i}, E_{C_i})$ has $|V_{C_i}| > 2^k$. Let V_1, \dots, V_p be the partition of V_{C_i} defined by the twin equivalence of $G[V_{C_i} \cup S]$. That is, two vertices $u, v \in V_{C_i}$ belong in a V_j , $j \in [p]$, if and only if u and v are twins. Note that $p \leq 2^k$. Observe that, since C_i is a clique, the graphs $C_i[V_j]$, $j \in [p]$, are also cliques. In other words, for each $j \in [p]$, all the vertices of V_j are adjacent twins. We delete all but one vertex of V_j , for each $j \in [p]$, and repeat this process for every $i \in [m]$ such that $|V_{C_i}| > 2^k$. Let $G' = (V', E')$ be the resulting subgraph of G and $d = |D|$, where D is the set of vertices that were removed throughout this process. It follows from Lemma 3 that $I_v(G) = I_v(G') + d$. Observe also that $S \subseteq V'$ and that each connected component of $G'[V' \setminus S]$ is a clique of at most 2^k vertices. In other words, G' has vertex integrity at most $2^k + k$. To sum up, to compute $I_v(G)$ it suffices to compute $I_v(G')$, which can be done in FPT-time by running the algorithm presented in Theorem 9. ◀

4 Edge-irregulators

In this section we begin the study of finding an optimal edge-irregulator of a given graph. It turns out that the decision version of this problem is \mathcal{NP} -complete, even for quite restrictive classes of graphs (see Theorem 12). Furthermore, it is also $W[1]$ -hard parameterised by the size of the solution.

► **Theorem 12** (*). *Let G be a graph and $k \in \mathbb{N}$. Deciding if $I_e(G) \leq k$ is \mathcal{NP} -complete, even if G is a planar bipartite graph of maximum degree 6.*

Sketch of Proof. The problem is clearly in \mathcal{NP} . We focus on showing it is also \mathcal{NP} -hard. This is achieved through a reduction from the PLANAR 3-SAT problem which is known to be \mathcal{NP} -complete [30]. In that problem, a 3CNF formula ϕ is given as an input. We say that a bipartite graph $G' = (V, C, E)$ corresponds to ϕ if it is constructed from ϕ in the following way: for each literal x_i (resp. $\neg x_i$) that appears in ϕ , add the *literal vertex* v_i (resp. v'_i) in V (for $1 \leq i \leq n$) and for each clause C_j of ϕ add a *clause vertex* c_j in C (for $1 \leq j \leq m$).



■ **Figure 2** The construction in the proof of Theorem 12. The dashed lines are used to represent the edges between the literal and the clause vertices.

Then the edge $v_i c_j$ (resp. $v'_i c_j$) is added if the literal x_i (resp. $\neg x_i$) appears in the clause C_j . Finally, we add the edge $v_i v'_i$ for every i . A 3CNF formula ϕ is valid as input to the PLANAR 3-SAT problem if the graph G' that corresponds to ϕ is planar. Furthermore, we may assume that each variable appears in ϕ twice as a positive and once as a negative literal [11]. The question is whether there exists a truth assignment to the variables of X satisfying ϕ . Starting from a 3CNF formula ϕ , we construct a graph G such that $I_e(G) \leq 3n$ if and only if ϕ is satisfiable.

Construction. We start with the graph G' that corresponds to the formula ϕ . Then, for each $1 \leq i \leq n$, we remove the edge $v_i v'_i$, and attach the gadget illustrated in Figure 2 to v_i and v'_i . Let E_i denote the edges of the gadget attached to v_i and v'_i plus the edges e_i^1, e_i^2 and e_i^3 . Finally, for each $1 \leq j \leq m$, we attach two leaves to c_j and then we add the star with 7 vertices and identify one of its leaves as the vertex c_j . Note that the edges e_i^1, e_i^2 correspond to the two positive appearances of the literal x_i , while the edge e_i^3 corresponds to the one negative appearance of the same literal. We stress that the edge e_i^4 is a “simple” edge leading to a vertex of degree 1, and does not correspond to any appearance (positive or negative) of x_i . Observe that the resulting graph G is planar, bipartite and $\Delta(G) = 6$. This finishes the construction.

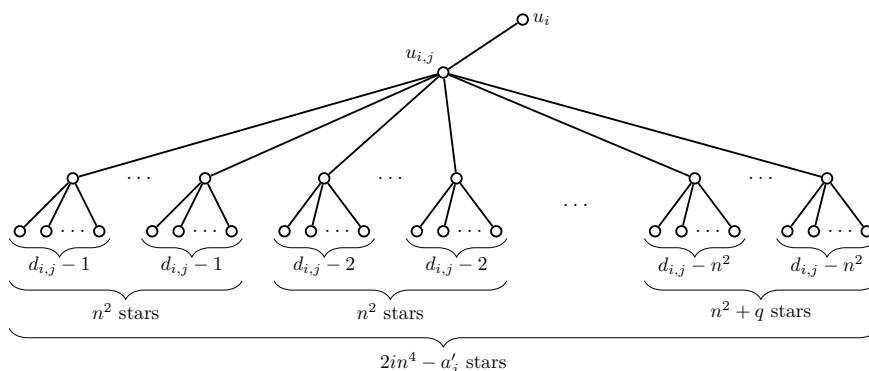
The rest of the reduction is based on some useful observations about the constructed graph. Indeed, we prove that for every edge-irregularator S of G , for every $1 \leq i \leq n$, we have that $|S \cap E_i| \geq 3$. Also, if S is such that $|S| \leq 3n$, then, for every $1 \leq i \leq n$, we have that if $|S \cap \{e_i^1, e_i^2\}| \geq 1$ then $|S \cap \{e_i^3, e_i^4\}| = 0$ and if $|S \cap \{e_i^3, e_i^4\}| \geq 1$ then $|S \cap \{e_i^1, e_i^2\}| = 0$. Finally, we show that there exists a satisfying truth assignment of ϕ if and only if $I_e(G) \leq 3n$, where G is the graph constructed from the input formula ϕ as explained above. Starting from a satisfying truth assignment of ϕ , we insert $\{v'_i u_6, e_i^1, e_i^2\}$ ($\{v_i u_6, e_i^3, e_i^4\}$ resp.) into the constructed edge-irregularator S if x_i is set to true (false resp.). For the reverse direction, we set the variable x_i to true if and only if $S \cap \{e_i^1, e_i^2\} \neq \emptyset$. ◀

► **Theorem 13** (★). *Let G be a graph and $k \in \mathbb{N}$. Deciding if $I_e(G) \leq k$ is W[1]-hard parameterised by k .*

The proof of this theorem is based on a reduction from k -MULTICOLOURED CLIQUE.

Additionally, this problem exhibits a similar behaviour to finding optimal vertex-irregularators, as it also remains intractable even for “relatively large” structural parameters.

► **Theorem 14.** *Let G and $k \in \mathbb{N}$. Deciding if $I_e(G) \leq k$ is W[1]-hard parameterised by either the feedback vertex set number or the treedepth of G .*



■ **Figure 3** The tree $T_{i,j}$ that is attached to the vertex u_i , where j is such that $a_j \in \bar{L}(u_i)$, in the proof of Theorem 14. The value of q is such that, in total, $d(u_{i,j}) = 2in^4 - a'_j + 1$.

Proof. The reduction is from the GENERAL FACTOR problem:

GENERAL FACTOR
Input: A graph $H = (V, E)$ and a list function $L : V \rightarrow 2^{\Delta(H)}$ that specifies the available degrees for each vertex $u \in V$.
Task: Does there exist a set $S \subseteq E$ such that $d_{H-S}(u) \in L(u)$ for all $u \in V$?

This problem is known to be W[1]-hard when parameterised by the vertex cover number of H [20].

Starting from an instance (H, L) of GENERAL FACTOR, we construct a graph G such that $I_e(G) \leq n^2$, where $n = |V(H)|$, if and only if (H, L) is a yes-instance. Moreover, the constructed graph G will have treedepth and feedback vertex set $\mathcal{O}(vc)$, where vc is the vertex cover number of H . For every vertex $u \in V(H)$, let us denote by $\bar{L}(u)$ the set $\{0, 1, \dots, d_H(u)\} \setminus L(u)$. In the case where $\{0, 1, \dots, d_H(u)\} \setminus L(u) = \emptyset$, we set $\bar{L}(u) = \{-1\}$. On a high level, the graph G is constructed by adding some trees on the vertices of H . In particular, for each vertex $u \in V(H)$ and for each element a in $\bar{L}(u)$, we will attach a tree to u whose purpose is to prevent u from having degree a in $G - S$, for any optimal edge-irregularator S of G . We proceed with the formal proof.

Construction. We begin by defining an arbitrary order on the vertices of H . That is, $V(H) = \{u_1, u_2, \dots, u_n\}$. Next, we describe the trees we will use in the construction of G . In particular, we will describe the trees that we attach to the vertex u_i , for every $1 \leq i \leq n$. First, for each $a_j \in \bar{L}(u_i)$, define the value $a'_j = d_H(u_i) - a_j$. Also, for each j , let $d_{i,j} = 2in^4 - a'_j$. For each “forbidden degree” a_j in the list $\bar{L}(u_i)$, we will attach a tree $T_{i,j}$ to u_i . We define the tree $T_{i,j}$ as follows.

First, for every $0 \leq k \leq n^2 - 1$, create n^2 copies of $S_{d_{i,j}-k}$ (the star on $d_{i,j} - k$ vertices) and q additional copies of $S_{d_{i,j}-n^2+1}$ (the exact value of q will be defined in what follows). Then, choose one leaf from each one of the above stars, and identify them into a single vertex denoted as $u_{i,j}$; the value of q is such that $d(u_{i,j}) = d_{i,j} - 1 = 2in^4 - a'_j - 1$. Let $T_{i,j}$ be the resulting tree and let us say that $u_{i,j}$ is the root of $T_{i,j}$ (see Figure 3).

Let us now describe the construction of G . For each vertex $u_i \in V(H)$ and for each $a_j \in \bar{L}(u_i)$, add the tree $T_{i,j}$ to H and the edge $u_{i,j}u_i$. Then, for each vertex $u_i \in V(H)$, for any j such that $u_{i,j}$ is a neighbour of u_i , add p_i additional copies of the tree $T_{i,j}$, as well as the edges between u_i and the roots of the additional trees, so that $d_G(u_i) = 2in^4$. The

18:12 Parameterised Distance to Local Irregularity

resulting graph is G . Note that, for each vertex of $V(H)$, we are adding at most $\mathcal{O}(n)$ trees, each one containing at most $\mathcal{O}(n^{10})$ vertices. Thus, the construction of G is achieved in polynomial time.

Reduction. Assume first that (H, L) is a yes-instance of GENERAL FACTOR, and let $S \subseteq E$ be such that $d_{H-S}(u) \in L(u)$ for all $u \in V(H)$. We claim that S is also an edge-irregulator of G . By the construction of G , and since S only contains edges from H , there are no two adjacent vertices in $G[V(G) \setminus V(H)]$ that have the same degree in $G - S$. Thus, it remains to check the pairs of adjacent vertices x, y such that, either both x and y belong to $V(H)$, or, w.l.o.g., $x \in V(H)$ and $y \in V(G - H)$. For the first case, let $x = u_i$ and $y = u_{i'}$, for $1 \leq i < i' \leq n$. Then, assuming that $d_{G-S}(u_i) = d_{G-S}(u_{i'})$, we get that $2in^4 - p = 2i'n^4 - p'$, where S contains $0 \leq p \leq n^2$ and $0 \leq p' \leq n^2$ edges incident to u_i and $u_{i'}$ respectively. Thus, $2n^4(i - i') = p - p'$, a contradiction since $-n^2 \leq p - p' \leq n^2$, $i < i'$ and $-n \leq i - i' \leq n$. For the second case, for every i , let $d_{G-S}(u_i) = 2in^4 - p$, where the set S contains $1 \leq p \leq n^2$ edges of H incident to u_i . Also, by the construction of G and since S only contains edges from H , we have that for every j , $d_{G-S}(u_{i,j}) = d_G(u_{i,j}) = 2in^4 - a'_j$, where, recall, $a'_j = d_H(u_i) - a_j$ for $a_j \in \bar{L}(u_i)$ (see Figure 3). Assume now that there exist i, j such that $d_{G-S}(u_i) = d_{G-S}(u_{i,j})$. Then, $2in^4 - p = 2in^4 - d_H(u_i) + a_j$ and thus $d_H(u_i) - p = a_j$. But then $d_{H-S}(u_i) = a_j$, which is a contradiction since $a_j \in \bar{L}(u_i)$. Thus, S is an edge-irregulator of G and $|S| \leq n^2$ since S only contains edges of $E(H)$.

For the reverse direction, assume that $I_e(G) \leq n^2$ and let S be an optimal edge-irregulator of G . We will show that S is also such that $d_{H-S}(u_i) \in L(u_i)$, for every i . Let us first prove the following claim.

▷ **Claim 15.** Let S be an optimal edge-irregulator of G . Then either $S \subseteq E(H)$ or $|S| > n^2$.

Proof. Assume there exist i, j such that $|S \cap E_{i,j}| = x \geq 1$ and $x \leq n^2$. Among those edges, there are $x_1 \geq 0$ edges incident to u and $x_2 \geq 0$ edges incident to children of u (but not to u), with $x_1 + x_2 = x \leq n^2$.

Assume first that $x_1 = 0$. Then $x = x_2$ and there is no edge of $S \cap E_{i,j}$ that is incident to u . Then $d_{G-S}(u) = d_G(u)$ and observe that $d_G(u)$ is strictly larger than that of any of its children (by the construction of G). It follows that $S \setminus E_{i,j}$ is also an edge-irregulator of G , contradicting the optimality of S . Thus $x_1 \geq 1$. It then follows from the construction of G that there exist at least n^2 children of u , denoted by z_1, \dots, z_{n^2} , such that $d_{G-S}(u) = d_G(z_k)$, for every $1 \leq k \leq n^2$. Since $x \leq n^2$, there exists at least one $1 \leq k \leq n^2$ such that $d_{G-S}(u) = d_{G-S}(z_k)$, contradicting the fact that S is an edge-irregulator. Thus $x > n^2$. ◁

It follows directly from Claim 15 that S contains only edges of $E(H)$. Assume that there exist i, j such that $d_{H-S}(u_i) = a_j$ and $a_j \in \bar{L}(u_i)$. Then $d_{G-S}(u_i) = 2in^4 - a'_j$. Also, by the construction of G , u_i is adjacent to a vertex $u_{i,j}$ for which (since S contains only edges of $E(H)$) we have that $d_{G-S}(u_{i,j}) = d_G(u_{i,j}) = 2in^4 - a'_j$. This is contradicting the fact that S is an edge-irregulator of G . Thus, for every i, j , we have that if $d_{H-S}(u_i) = a_j$, then $a_j \in L(u_i)$, which finishes our reduction.

Finally, if H has vertex cover number vc , then, by Observation 4, we have that G has treedepth and feedback vertex set $\mathcal{O}(vc)$. ◀

We close this section by observing that the proof of Theorem 9 can be adapted for the case of edge-irregulators. Indeed, it suffices to replace the guessing of vertices and the variables defined on vertices, by guessing of edges and variables defined on the edges of the given graph. Finally, the definition of the sub-types is done through subgraphs produced only by deletion of edges. This leads us to the following:

► **Corollary 16.** *Given a graph G with vertex integrity k , there exists an algorithm that computes $I_e(G)$ in FPT-time parameterised by k .*

5 Conclusion

In this work we continued the study of the problem of finding optimal vertex-irregularators, and introduced the problem of finding optimal edge-irregularators. In the case of vertex-irregularators, our results are somewhat optimal, in the sense that we almost characterise which are the “smallest” graph-structural parameters that render this problem tractable. The only “meaningful” parameter whose behaviour remains unknown is the modular-width of the input graph. The parameterised behaviour of the case of edge-irregularators is also somewhat understood, but there are still some parameters for which the problem remains open. Another interesting direction is that of approximating optimal vertex or edge-irregularators. In particular it would be interesting to identify parameters for which either problem becomes approximable in FPT-time (recall that vertex-irregularators are not approximable within any decent factor in polynomial time [16]). Finally, provided that the behaviour of edge-irregularators is better understood, we would also like to propose the problem of finding locally irregular minors, of maximum order, of a given graph G .

References

- 1 Agostinho Agra, Geir Dahl, Torkel Andreas Haufmann, and Sofia J. Pinheiro. The k -regular induced subgraph problem. *Discrete Applied Mathematics*, 222:14–30, 2017. doi:10.1016/j.dam.2017.01.029.
- 2 Noga Alon, Michael Krivelevich, and Wojciech Samotij. Largest subgraph from a hereditary property in a random graph. *Discrete Mathematics*, 346(9):113480, 2023. doi:10.1016/J.DISC.2023.113480.
- 3 Yuichi Asahiro, Hiroshi Eto, Takehiro Ito, and Eiji Miyano. Complexity of finding maximum regular induced subgraphs with prescribed degree. *Theoretical Computer Science*, 550:21–35, 2014. doi:10.1016/j.tcs.2014.07.008.
- 4 Olivier Baudon, Julien Bensmail, Jakub Przybyło, and Mariusz Woźniak. On decomposing regular graphs into locally irregular subgraphs. *European Journal of Combinatorics*, 49:90–104, 2015. doi:10.1016/j.ejc.2015.02.031.
- 5 Rémy Belmonte and Ignasi Sau. On the complexity of finding large odd induced subgraphs and odd colorings. *Algorithmica*, 83(8):2351–2373, 2021. doi:10.1007/s00453-021-00830-x.
- 6 Julien Bensmail. Maximising 1’s through Proper Labellings. Research report, Côte d’Azur University, 2023. URL: <https://hal.science/hal-04086726>.
- 7 Robert Brederick, Sepp Hartung, André Nichterlein, and Gerhard J. Woeginger. The complexity of finding a large subgraph under anonymity constraints. In *Algorithms and Computation - 24th International Symposium, ISAAC 2013*, volume 8283 of *Lecture Notes in Computer Science*, pages 152–162. Springer, 2013. doi:10.1007/978-3-642-45030-3_15.
- 8 Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996. doi:10.1016/0020-0190(96)00050-6.
- 9 Gruia Călinescu, Cristina G. Fernandes, Hemanshu Kaul, and Alexander Zelikovsky. Maximum series-parallel subgraph. *Algorithmica*, 63(1-2):137–157, 2012. doi:10.1007/S00453-011-9523-4.
- 10 Markus Chimani, Ivo Hedtke, and Tilo Wiedera. Exact algorithms for the maximum planar subgraph problem: New models and experiments. *ACM Journal of Experimental Algorithmics*, 24(1):2.1:1–2.1:21, 2019. doi:10.1145/3320344.

- 11 Elias Dahlhaus, David S. Johnson, Christos H. Papadimitriou, Paul D. Seymour, and Mihalis Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, 23(4):864–894, 1994. doi:10.1137/S0097539792225297.
- 12 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012. doi:10.1007/978-3-662-53622-3.
- 13 Pål Grønås Drange, Markus S. Dregi, and Pim van 't Hof. On the computational complexity of vertex integrity and component order connectivity. *Algorithmica*, 76(4):1181–1202, 2016. doi:10.1007/S00453-016-0127-X.
- 14 Baris Can Esmer, Ariel Kulik, Dániel Marx, Daniel Neuen, and Roohani Sharma. Faster exponential-time approximation algorithms using approximate monotone local search. In *30th Annual European Symposium on Algorithms, ESA 2022*, volume 244 of *LIPICs*, pages 50:1–50:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ESA.2022.50.
- 15 Michael R. Fellows, Fedor V. Fomin, Daniel Lokshtanov, Frances A. Rosamond, Saket Saurabh, Stefan Szeider, and Carsten Thomassen. On the complexity of some colorful problems parameterized by treewidth. *Information and Computation*, 209(2):143–153, 2011. doi:10.1016/J.IC.2010.11.026.
- 16 Foivos Fioravantes, Nikolaos Melissinos, and Theofilos Triommatis. Complexity of finding maximum locally irregular induced subgraphs. In *18th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2022*, volume 227 of *LIPICs*, pages 24:1–24:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.SWAT.2022.24.
- 17 Fedor V. Fomin, Serge Gaspers, Daniel Lokshtanov, and Saket Saurabh. Exact algorithms via monotone local search. *Journal of the ACM*, 66(2):8:1–8:23, 2019. doi:10.1145/3284176.
- 18 Robert Ganian, Fabian Klute, and Sebastian Ordyniak. On structural parameterizations of the bounded-degree vertex deletion problem. *Algorithmica*, 83(1):297–336, 2021. doi:10.1007/S00453-020-00758-8.
- 19 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- 20 Gregory Z. Gutin, Eun Jung Kim, Arezou Soleimanfallah, Stefan Szeider, and Anders Yeo. Parameterized complexity results for general factors in bipartite graphs with an application to constraint programming. *Algorithmica*, 64(1):112–125, 2012. doi:10.1007/S00453-011-9548-8.
- 21 Falk Hüffner, Christian Komusiewicz, Hannes Moser, and Rolf Niedermeier. Fixed-parameter algorithms for cluster vertex deletion. *Theory of Computing Systems*, 47(1):196–217, 2010. doi:10.1007/S00224-008-9150-X.
- 22 Mark Jones, Daniel Lokshtanov, M. S. Ramanujan, Saket Saurabh, and Ondrej Suchý. Parameterized complexity of directed steiner tree on sparse graphs. *SIAM J. Discret. Math.*, 31(2):1294–1327, 2017. doi:10.1137/15M103618X.
- 23 Michał Karoński, Tomasz Łuczak, and Andrew Thomason. Edge weights and vertex colors. *Journal of Combinatorial Theory*, 91:151–157, May 2004. doi:10.1016/j.jctb.2003.12.001.
- 24 Ralph Keusch. A solution to the 1-2-3 conjecture. *Journal of Combinatorial Theory, Series B*, 166:183–202, 2024. doi:10.1016/J.JCTB.2024.01.002.
- 25 Subhash Khot and Venkatesh Raman. Parameterized complexity of finding subgraphs with hereditary properties. *Theoretical Computer Science*, 289(2):997–1008, 2002. doi:10.1016/S0304-3975(01)00414-5.
- 26 Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, 2012. doi:10.1007/S00453-011-9554-X.
- 27 Michael Lampis and Manolis Vasilakis. Structural parameterizations for two bounded degree problems revisited. In *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPICs*, pages 77:1–77:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.ESA.2023.77.
- 28 H. W. Lenstra Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983. doi:10.1287/MOOR.8.4.538.

- 29 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is np-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4.
- 30 David Lichtenstein. Planar formulae and their uses. *SIAM journal on computing*, 11(2):329–343, 1982. doi:10.1137/0211025.
- 31 Hannes Moser and Dimitrios M. Thilikos. Parameterized complexity of finding regular induced subgraphs. *Journal of Discrete Algorithms*, 7(2):181–190, 2009. doi:10.1016/j.jda.2008.09.005.
- 32 J. Barkley Rosser and Lowell Schoenfeld. Approximate formulas for some functions of prime numbers. *Illinois Journal of Mathematics*, 6(1):64–94, 1962.
- 33 A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer-Verlag Berlin and Heidelberg GmbH & Co., 2003.

Single-Machine Scheduling to Minimize the Number of Tardy Jobs with Release Dates

Matthias Kaul¹  

Universität Bonn, Bonn, Germany

Matthias Mnich  

Hamburg University of Technology, Institute for Algorithms and Complexity, Hamburg, Germany

Hendrik Molter  

Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Abstract

We study the fundamental scheduling problem $1 \mid r_j \mid \sum w_j U_j$: schedule a set of n jobs with weights, processing times, release dates, and due dates on a single machine, such that each job starts after its release date and we maximize the weighted number of jobs that complete execution before their due date. Problem $1 \mid r_j \mid \sum w_j U_j$ generalizes both KNAPSACK and PARTITION, and the simplified setting without release dates was studied by Hermelin et al. [Annals of Operations Research, 2021] from a parameterized complexity viewpoint.

Our main contribution is a thorough complexity analysis of $1 \mid r_j \mid \sum w_j U_j$ in terms of four key problem parameters: the number $p_\#$ of processing times, the number $w_\#$ of weights, the number $d_\#$ of due dates, and the number $r_\#$ of release dates of the jobs. $1 \mid r_j \mid \sum w_j U_j$ is known to be weakly para-NP-hard even if $w_\# + d_\# + r_\#$ is constant, and Heeger and Hermelin [ESA, 2024] recently showed (weak) W[1]-hardness parameterized by $p_\#$ or $w_\#$ even if $r_\#$ is constant.

Algorithmically, we show that $1 \mid r_j \mid \sum w_j U_j$ is fixed-parameter tractable parameterized by $p_\#$ combined with any two of the remaining three parameters $w_\#$, $d_\#$, and $r_\#$. We further provide pseudo-polynomial XP-time algorithms for parameter $r_\#$ and $d_\#$. To complement these algorithms, we show that $1 \mid r_j \mid \sum w_j U_j$ is (strongly) W[1]-hard when parameterized by $d_\# + r_\#$ even if $w_\#$ is constant. Our results provide a nearly complete picture of the complexity of $1 \mid r_j \mid \sum w_j U_j$ for $p_\#$, $w_\#$, $d_\#$, and $r_\#$ as parameters, and extend those of Hermelin et al. [Annals of Operations Research, 2021] for the problem $1 \parallel \sum w_j U_j$ without release dates.

2012 ACM Subject Classification Theory of computation \rightarrow Fixed parameter tractability; Theory of computation \rightarrow Scheduling algorithms

Keywords and phrases Scheduling, Release Dates, Fixed-Parameter Tractability

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.19

Related Version *Full Version*: <https://arxiv.org/abs/2408.12967> [16]

Funding *Matthias Mnich*: Partially supported by DFG project MN 59/4-1.

Hendrik Molter: Supported by the European Union’s Horizon Europe research and innovation programme under grant agreement 949707.

Acknowledgements We wish to thank Danny Hermelin and Dvir Shabtay for fruitful discussions that led to some of the results of this work.

¹ Most work while affiliated with Hamburg University of Technology, Institute for Algorithms and Complexity, Hamburg, Germany.



1 Introduction

The problem of scheduling jobs to machines is one of the core application areas of combinatorial optimization [25]. Typically, the task is to allocate jobs to machines in order to minimize a certain objective function while complying with certain constraints. In our setting, the jobs are characterized by several numerical parameters: a *processing time*, a *release date*, a *due date*, and a *weight*. We have access to a single machine that can process one job (non-preemptively) at a time. We consider one of the most fundamental objective functions, namely to minimize the weighted number of tardy jobs, where a job is considered *tardy* if it completes after its due date. In the standard three-field notation for scheduling problems by Graham [8], the problem is called $1 \mid r_j \mid \sum w_j U_j$. We give a formal definition in Section 2.

The interest in $1 \mid r_j \mid \sum w_j U_j$ comes from various sources. It generalizes several fundamental combinatorial problems. In the most simple setting, without weights and release dates, the classic algorithm by Moore [24] computes an optimal schedule in polynomial time. When weights are added, the problem ($1 \parallel \sum w_j U_j$) encapsulates the KNAPSACK problem, a cornerstone in combinatorial optimization and one of Karp’s 21 NP-complete problems [15]. Precisely, when all jobs are released at time zero and all jobs have a common due date, we obtain the KNAPSACK problem. Karp’s NP-hardness proof (from his seminal paper [15]) is the first example of a reduction to a problem involving numbers. The problem $1 \parallel \sum w_j U_j$ is one of the most extensively studied problems in scheduling and can be solved in pseudopolynomial time by the classic algorithm by Lawler and Moore [19]. Hermelin et al. [14] showed that this algorithm can be improved in various restricted settings. Better running times have been achieved for the special case where the weights of the jobs equal their processing times [3, 6, 17, 26].

The problem $1 \parallel \sum w_j U_j$ (so without release dates) has been studied from the perspective of parameterized complexity by Hermelin et al. [13]. They considered the number $p_{\#}$ of different processing times, the number $d_{\#}$ of different due dates, and the number $w_{\#}$ of different weights as parameters and showed fixed-parameter tractability for $w_{\#} + p_{\#}$, $w_{\#} + d_{\#}$, and $p_{\#} + d_{\#}$ as well as giving an XP-algorithm for the parameters $p_{\#}$ and $w_{\#}$. These results are presumably tight, as Heeger and Hermelin [9] recently showed (weak) W[1]-hardness for the parameters $p_{\#}$ and $w_{\#}$. The problem has also been studied under fairness aspects [10].

The addition of release dates is naturally motivated in every scenario where not all jobs are initially available. The aim of this paper is to study the parameterized complexity of the problem ($1 \mid r_j \mid \sum w_j U_j$) in this setting. Here, it encapsulates PARTITION and becomes weakly NP-hard [20], even if there are only two different release dates and two different due dates and all jobs have the same weight. It has previously been studied for the case of uniform processing times, both on a single machine [7] and for parallel machines [2, 11], as well as for the special case of interval scheduling [1, 12, 18, 27].

Our Contributions. In this paper, we deploy the tools of parameterized complexity to study the computational complexity of $1 \mid r_j \mid \sum w_j U_j$. In the spirit of “parameterizing by the number of numbers” [5], we analyze the complexity picture with respect to (i) the number $p_{\#}$ of distinct processing times of the jobs, (ii) the number $w_{\#}$ of distinct weights of the jobs, (iii) the number $d_{\#}$ of distinct due dates of the jobs, and (iv) the number $r_{\#}$ of distinct release dates of the jobs. Thereby, we extend and complement the results obtained by Hermelin et al. [13] for the case where all release dates are zero.

In summary, we obtain an almost complete classification into tractable cases (meaning that we find a fixed-parameter algorithm) and intractable cases (meaning that we show the problem to be W[1]-hard) depending on which subset of parameters from $\{p_{\#}, w_{\#}, d_{\#}, r_{\#}\}$ we consider.

First note that for some parameter combinations, the problem complexity has already been resolved. In particular, $1 \mid r_j \mid \sum w_j U_j$ is known to be weakly NP-hard for $d_{\#} = 1$ and $r_{\#} = 1$, as this setting captures the KNAPSACK problem [15]. Furthermore, $1 \mid r_j \mid \sum w_j U_j$ is known to be weakly NP-hard for $d_{\#} = 2$, $w_{\#} = 1$, and $r_{\#} = 2$ [20]. For parameter $p_{\#}$ as well as $w_{\#}$, Heeger and Hermelin [9] showed weak W[1]-hardness even if $r_{\#} = 1$.

That leaves open the parameterized complexity for several parameter combinations. We extend the known hardness results by showing the following:

- $1 \mid r_j \mid \sum w_j U_j$ is (strongly) W[1]-hard parameterized by $d_{\#} + r_{\#}$ even if $w_{\#} = 1$. This result is obtained by a straightforward reduction showing that $1 \mid r_j \mid \sum w_j U_j$ generalizes BIN PACKING. Our main results are on the algorithmic side, where we show the following:
- $1 \mid r_j \mid \sum w_j U_j$ is fixed-parameter tractable parameterized by $p_{\#} + d_{\#} + r_{\#}$.
- $1 \mid r_j \mid \sum w_j U_j$ is fixed-parameter tractable parameterized by $p_{\#} + w_{\#} + d_{\#}$ or $p_{\#} + w_{\#} + r_{\#}$.
- $1 \mid r_j \mid \sum w_j U_j$ can be solved in pseudo-polynomial time for constant $r_{\#}$ or constant $d_{\#}$.

For the first two, we employ reductions to MIXED INTEGER LINEAR PROGRAMMING (MILP), and for the latter, we give a dynamic programming algorithm. Due to space constraints, proofs of results marked with \star are deferred to a full version [16].

With our results, we resolve the parameterized complexity of $1 \mid r_j \mid \sum w_j U_j$ for almost all parameter combinations of $\{p_{\#}, w_{\#}, r_{\#}, d_{\#}\}$ and hence give the first comprehensive overview thereof. The remaining question is whether $1 \mid r_j \mid \sum w_j U_j$ is polynomial-time solvable for *constant* $p_{\#}$ or fixed-parameter tractable for $p_{\#} + w_{\#}$. It also remains open whether $1 \mid r_j \mid \sum w_j U_j$ is fixed-parameter tractable parameterized by $p_{\#}$ if all numbers are encoded in unary. A technical report [4] claims (strong) NP-hardness even for $p_{\#} = 2$ and $w_{\#} = 1$. If that result holds, then it would also settle the parameterized complexity for the parameter combination $p_{\#} + w_{\#}$, and for $p_{\#}$ when all processing times and weights are encoded in unary. Otherwise, those questions would remain open.

Our results contribute to the growing body of investigating the parameterized complexity of fundamental scheduling problems [23]; for reference, we refer to the open problem collection by Mnich and van Bevern [22].

2 Preliminaries

Scheduling. The problem considered in this work is denoted $1 \mid r_j \mid \sum w_j U_j$ in the standard three-field notation for scheduling problems by Graham [8]. In this problem, we have n jobs and one machine that can process one job at a time. Each job $j \in \{1, \dots, n\}$ has a *processing time* p_j , a *release date* r_j , a *due date* d_j , and a *weight* w_j , where we p_j , r_j , d_j , and w_j are non-negative integers. We use $p_{\#}$, $r_{\#}$, $d_{\#}$, and $w_{\#}$ to denote the number of different processing times, release dates, due dates, and weights, respectively.

A *schedule* $\sigma : \{1, \dots, n\} \rightarrow \mathbb{N}$ assigns to each job j a *starting time* $\sigma(j)$ to process it until its *completion time* $\sigma(j) + p_j$, so no other job $j' \neq j$ must start during j 's *execution time* $\sigma(j), \dots, \sigma(j) + p_j - 1$. We call a job j *early* in a schedule σ if $\sigma(j) + p_j \leq d_j$; otherwise we call job j *tardy*. We say that the machine is *idle* at time s in a schedule σ if no job's execution time contains s . The goal is to find a schedule that minimizes the weighted number of tardy jobs or, equivalently, maximizes the weighted number of early jobs

$$W = \sum_{j \mid (\sigma(j) = s \wedge s + p_j \leq d_j)} w_j .$$

19:4 Single-Machine Scheduling to Minimize the Number of Tardy Jobs

We call a schedule that maximizes the weighted number of early jobs *optimal*. Formally, the problem is defined as follows:

$$1 \mid r_j \mid \sum w_j U_j$$

Input: A number n of jobs, a list of processing times (p_1, p_2, \dots, p_n) , a list of release dates (r_1, r_2, \dots, r_n) , a list of due dates (d_1, d_2, \dots, d_n) , and a list of weights (w_1, w_2, \dots, w_n) .

Task: Compute an optimal schedule, that is, a schedule σ that maximizes $W = \sum_{j \mid \sigma(j) = s \wedge s + p_j \leq d_j} w_j$.

Given an instance I of $1 \mid r_j \mid \sum w_j U_j$, we make the following observation:

► **Observation 1.** *Let I be an instance of $1 \mid r_j \mid \sum w_j U_j$ and let d_{\max} be the largest due date of any job in I . Let I' be the instance obtained from I by setting $r'_j = d_{\max} - d_j$ and $d'_j = d_{\max} - r_j$ for each job j . Then I admits a schedule where the weighted number of early jobs is W if and only if I' admits a schedule where the weighted number of early jobs is W .*

Observation 1 holds, as we can transform a schedule σ for I into a schedule σ' for I' (with the same weighted number of early jobs) by setting $\sigma'(j) = d_{\max} - \sigma(j) - p_j$. Intuitively, this means that we can switch the roles of release dates and due dates to obtain instances with the same objective value.

Mixed Integer Linear Programming. For several of our algorithmic results, we use reductions to MIXED INTEGER LINEAR PROGRAMMING (MILP). This problem is defined as follows:

MIXED INTEGER LINEAR PROGRAMMING (MILP)

Input: A vector x of n variables, a subset S of the variables which are considered integer variables, a constraint matrix $A \in \mathbb{R}^{m \times n}$, and two vectors $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$.

Task: Compute an assignment to the variables (if one exists) such that all integer variables in S are set to integer values, $Ax \leq b$, $x \geq 0$, and $c^T x$ is maximized.

If all variables are integer variables, the problem is simply called INTEGER LINEAR PROGRAMMING (ILP). Due to Lenstra's well-known result for MILP [21], we have that:

► **Theorem 2** ([21]). *MILP is fixed-parameter tractable when parameterized by the number of integer variables.*

3 Hardness Results

In this section, we first discuss known hardness results for $1 \mid r_j \mid \sum w_j U_j$, and then present a novel parameterized hardness result. Observe that for $r_j = 0$ and $d_j = d$ (that is, all jobs have the same deadline), the problem $1 \mid r_j \mid \sum w_j U_j$ is equivalent to KNAPSACK, which is known to be weakly NP-hard [15]. Further, there is a straightforward reduction from PARTITION to $1 \mid r_j \mid \sum w_j U_j$ that only uses two release dates and two due dates, and uniform weights [20]. Finally, Heeger and Hermelin [9] recently showed that the special case of $1 \mid r_j \mid \sum w_j U_j$ without release dates is weakly W[1]-hard when parameterized by either $p_{\#}$ or $w_{\#}$. Hence, (together with Observation 1) we have that:

- **Proposition 3** ([9, 15, 20]). *The problem $1 \mid r_j \mid \sum w_j U_j$ is*
- *weakly NP-hard even if $d_{\#} = 1$ and $r_{\#} = 1$,*
 - *weakly NP-hard even if $r_{\#} = d_{\#} = 2$ and $w_{\#} = 1$,*
 - *and weakly W[1]-hard when parameterized by either $p_{\#}$ or $w_{\#}$ even if $r_{\#} = 1$ or if $d_{\#} = 1$.*

The reduction from PARTITION to $1 \mid r_j \mid \sum w_j U_j$ by Lenstra et al. [20] can straightforwardly be extended to a reduction from BIN PACKING, which yields the following result:

► **Theorem 4** (★). *The problem $1 \mid r_j \mid \sum w_j U_j$ is strongly NP-hard, and strongly W[1]-hard when parameterized by $r_{\#} + d_{\#}$, even if $w_{\#} = 1$.*

4 $1 \mid r_j \mid \sum w_j U_j$ parameterized by $p_{\#} + r_{\#} + d_{\#}$

In this section, we present the following result.

► **Theorem 5**. *The problem $1 \mid r_j \mid \sum w_j U_j$ is fixed-parameter tractable when parameterized by $p_{\#} + r_{\#} + d_{\#}$.*

To prove Theorem 5, we present a reduction from $1 \mid r_j \mid \sum w_j U_j$ to MILP that creates instances of MILP where the number of integer variables is upper-bounded by a function of $p_{\#}$, $r_{\#}$, and $d_{\#}$. The result then follows from Theorem 2.

Given an instance of $1 \mid r_j \mid \sum w_j U_j$, we say that two jobs j and j' have the same type if they have the same processing time, the same release date, and the same due date, that is, $p_j = p_{j'}$, $r_j = r_{j'}$, and $d_j = d_{j'}$. Let \mathcal{T} denote the set of all types. Note that we have $|\mathcal{T}| \leq p_{\#} \cdot r_{\#} \cdot d_{\#}$. Furthermore, we sort all release dates and due dates such that if the k^{th} release date equals the ℓ^{th} due date, we require the release date to appear later in the ordering. Let \mathcal{L} denote an ordered list of all release dates and due dates that complies to the aforementioned requirement. Note that we have $|\mathcal{L}| \leq r_{\#} + d_{\#}$.

We now create an integer variable $x_{a,b}^t$ for all $t \in \mathcal{T}$ and all $a, b \in \mathcal{L}$ with $a < b$. Intuitively, if a and b are consecutive in \mathcal{L} , then this variable tells us how many jobs of type t to schedule in the time interval $[a, b]$. If a and b are not consecutive in \mathcal{L} , then $x_{a,b}^t$ is a zero-one variable that tells us whether we schedule a job of type t in a way such that its processing time intersects all $c \in \mathcal{L}$ with $a < c < b$.

We create the following constraints. The first set of constraints is

$$\forall t \in \mathcal{T} : \sum_{a,b \in \mathcal{L} \mid a < b} x_{a,b}^t \leq n_t, \quad (1)$$

where n_t denotes the number of jobs of type t in the $1 \mid r_j \mid \sum w_j U_j$ instance. Intuitively, these constraints ensure that we do not try to schedule more jobs of type t than there are available.

The second set of constraints is

$$\forall a, b \in \mathcal{L} \text{ with } a < b \text{ and } \forall t \in \mathcal{T} \text{ with } r_t > b \text{ or } d_t < a : x_{a,b}^t = 0, \quad (2)$$

where r_t denotes the release date of jobs of type t and d_t denotes the due date of jobs of type t . Intuitively, these constraints prevent us from trying to schedule a job of a certain type into an interval that conflicts with the job's release date or due date.

The third set of constraints is

$$\forall c \in \mathcal{L} : \sum_{t \in \mathcal{T}} \sum_{a,b \in \mathcal{L} \mid a < c < b} x_{a,b}^t \leq 1. \quad (3)$$

Intuitively, these constraints ensure that for each $c \in \mathcal{L}$ at most one job is scheduled that intersects c .

19:6 Single-Machine Scheduling to Minimize the Number of Tardy Jobs

The fourth set of constraints is

$\forall a, b \in \mathcal{L}$ with $a < b$:

$$\sum_{t \in \mathcal{T}} \left(\sum_{a', b' \in \mathcal{L} | a' \geq a, b' \leq b, a' < b'} p_t \cdot x_{a', b'}^t + \sum_{a', b' \in \mathcal{L} | a' < a, b' > b} (b - a) \cdot x_{a', b'}^t \right) \leq b - a, \quad (4)$$

where p_t denotes the processing time of jobs of type t . Intuitively, these constraints make sure that for every interval $[a, b]$ with $a, b \in \mathcal{L}$ we do not schedule jobs with total processing time more than $b - a$ into that interval.

Now we specify the objective function. If we want to schedule a certain number of jobs with type t early, we take the ones with the largest weight in order to minimize the weighted number of tardy jobs. Let $x^t = \sum_{a, b \in \mathcal{L}; a < b} x_{a, b}^t$. Intuitively, x^t is the number of jobs with type t that are scheduled early. For each type, $t \in \mathcal{T}$, order the jobs of type t in the $1 \mid r_j \mid \sum w_j U_j$ instance by their weight (largest to smallest) and let w_i^t denote the i^{th} largest weight of jobs with type t . The objective function we aim to maximize is

$$\sum_{t \in \mathcal{T}} \sum_{i=1}^{x^t} w_i^t. \quad (5)$$

Note that constraints (1), (2), (3), and (4) are linear. The objective function (5) is convex [13] and it is known that we can obtain an equivalent MILP with a linear objective function at the cost of introducing additional fractional variables and constraints [13]. Furthermore, we can observe the following:

► **Observation 6.** *The number of integer variables in the created MILP instance is in $O(p_{\#} \cdot r_{\#} \cdot d_{\#} \cdot (r_{\#} + d_{\#})^2)$.*

Next, we show the correctness of the reduction.

► **Lemma 7** (\star). *If the $1 \mid r_j \mid \sum w_j U_j$ instance admits a schedule where the weighted number of early jobs is W , then the created MILP instance admits a feasible solution that has objective value at least W .*

► **Lemma 8.** *If the created MILP instance is feasible and admits a solution with objective value W , then the $1 \mid r_j \mid \sum w_j U_j$ instance admits a schedule where the weighted number of early jobs is at least W .*

Proof. Suppose we are given a solution to the MILP instance with objective value W . We create a schedule σ as follows.

We iterate through pairs $a, b \in \mathcal{L}$ with $a < b$ as follows. We start with the smallest element $a \in \mathcal{L}$ and the smallest element $b \in \mathcal{L}$ with $a < b$ according to the ordering. We maintain a current starting point s that is initially set to $s = a$. Furthermore, we consider all jobs initially as “unscheduled”. We proceed as follows.

- We iterate through all $t \in \mathcal{T}$ in some arbitrary but fixed way. If $x_{a, b}^t > 0$, take the $x_{a, b}^t$ unscheduled jobs of type t with the maximum weight and schedule those between s and $s + x_{a, b}^t \cdot p_t$, where p_t is the processing time of jobs of type t . Now consider those jobs scheduled and set $s \leftarrow s + x_{a, b}^t \cdot p_t$. Continue with the next type.
- Replace b with the next-larger element in \mathcal{L} . If b is the largest element in \mathcal{L} , then replace a with the next-larger element in \mathcal{L} , set b to the smallest element $b \in \mathcal{L}$ with $a < b$ according to the ordering, and set $s \leftarrow \max\{a, s\}$. If a is the largest element of \mathcal{L} , terminate the process. Otherwise, go to the first step.

Since the solution to the MILP obeys constraint (1), we have that there are sufficiently many jobs of each type that can be scheduled. All jobs that remained unscheduled after the above-described procedure are scheduled in some arbitrary but feasible way.

We claim that the above procedure produces a schedule where the weighted number of early jobs is at least W . We start by showing that the procedure indeed produces a schedule. Clearly, there are no two jobs in conflict in the produced schedule. Furthermore, since we always have $s \geq a$ and since the solution to the MILP obeys constraints (2), we have that no job is scheduled to start before their release date.

In the remainder of the proof, we show that we schedule $x^t = \sum_{a,b \in \mathcal{L}; a < b} x_{a,b}^t$ jobs of type t early. In particular, we schedule the x^t jobs of type t with the largest weights early. As the solution to the MILP has objective value W , it follows that the total weight of early jobs is at least W .

We show that all jobs scheduled in the first step of the above-described procedure are early. To this end, we identify where the procedure inserts idle times and argue that all jobs scheduled in the first step between two consecutive idle times by the procedure are early. An idle time is inserted by the procedure whenever we set $s \leftarrow \max\{a, s\}$ and we have $a > s$. Consider the case where we set $s \leftarrow \max\{a, s\} = a$ for some $a \in \mathcal{L}$. (Note that, for technical reasons, this includes the case where $a = s$.) Let $a' \in \mathcal{L}$ denote the next larger element of \mathcal{L} for which we set $s \leftarrow \max\{a', s\} = a'$. Now suppose, for the sake of contradiction, that there is a job with a starting time between a and a' that is scheduled in the first step of the procedure and is tardy. Let j be the first such job, that is, the one with the smallest starting time. Let $x_{a'',b}^t \geq 1$ with some $a \leq a'' \leq a'$ be the variable that was considered by the procedure when j was scheduled. Then we have that $d_j \geq b$, since otherwise constraints (2) are not met. We make the following case distinction:

1. If $a = a''$, then the completion time of j is at most $a + \sum_{t \in \mathcal{T}} \sum_{b' \in \mathcal{L} | b' \leq b, a < b'} p_t \cdot x_{a,b'}^t$. However, since constraints (4) are met by the solution to the MILP, in particular the one for $a, b \in \mathcal{L}$, we have that

$$a + \sum_{t \in \mathcal{T}} \sum_{b' \in \mathcal{L} | b' \leq b, a < b'} p_t \cdot x_{a,b'}^t \leq b .$$

Since $b \leq d_j$, this contradicts the assumption that j is tardy.

2. Assume that $a < a'' \leq a'$. We argue that in this case, for all $x_{a''',b'}^t$ with $t' \in \mathcal{T}$ and $a''', b' \in \mathcal{L}$ with $a \leq a''' < a''$ and $b' > b$ we must have that $x_{a''',b'}^t = 0$. Assume that $x_{a''',b'}^t$ for some $t' \in \mathcal{T}$ and $a''', b' \in \mathcal{L}$ with $a \leq a''' < a''$ and $b' > b$. Consider the constraint (4) for $a'', b \in \mathcal{L}$. Then we have

$$\sum_{t \in \mathcal{T}} \left(\sum_{a', b' \in \mathcal{L} | a' \geq a'', b' \leq b, a' < b'} p_t \cdot x_{a',b'}^t + \sum_{a', b' \in \mathcal{L} | a' < a'', b' > b} (b - a'') \cdot x_{a',b'}^t \right) \leq b - a'' .$$

However, we also have that

$$\begin{aligned} b - a'' &< p_t \cdot x_{a'',b}^t + (b - a'') \cdot x_{a''',b'}^t \\ &\leq \sum_{t \in \mathcal{T}} \left(\sum_{a', b' \in \mathcal{L} | a' \geq a'', b' \leq b, a' < b'} p_t \cdot x_{a',b'}^t + \sum_{a', b' \in \mathcal{L} | a' < a'', b' > b} (b - a'') \cdot x_{a',b'}^t \right), \end{aligned}$$

contradicting the assumption that $x_{a''',b'}^t \geq 1$. It follows that the completion time of job j is at most $a + \sum_{t \in \mathcal{T}} \sum_{a', b' \in \mathcal{L} | a' \geq a, b' \leq b, a' < b'} p_t \cdot x_{a',b'}^t$. Since constraint (4) is met for $a, b \in \mathcal{L}$, we have that

$$a + \sum_{t \in \mathcal{T}} \sum_{a', b' \in \mathcal{L} | a' \geq a, b' \leq b, a' < b'} p_t \cdot x_{a',b'}^t \leq b .$$

19:8 Single-Machine Scheduling to Minimize the Number of Tardy Jobs

Since $b \leq d_j$, this contradicts the assumption that j is tardy. We conclude that all jobs scheduled by the above-described procedure in the first step are early. Since for every $a, b \in \mathcal{L}$ and $t \in \mathcal{T}$, the procedure schedules the $x_{a,b}^t$ jobs of type t with the largest weights early, we have that the total weight of early jobs is at least W . This concludes the proof. \blacktriangleleft

Now we have all the pieces available that are needed to prove Theorem 5.

Proof of Theorem 5. Theorem 5 follows directly from Observation 6, Lemmas 7 and 8, and Theorem 2. \blacktriangleleft

5 $1 \mid r_j \mid \sum w_j U_j$ parameterized by $p_{\#} + w_{\#} + d_{\#}$

In this section, we present the following result.

► **Theorem 9.** *The problem $1 \mid r_j \mid \sum w_j U_j$ is fixed-parameter tractable when parameterized by $p_{\#} + w_{\#} + r_{\#}$ or when parameterized by $p_{\#} + w_{\#} + d_{\#}$.*

We show the second part of Theorem 9, that is, $1 \mid r_j \mid \sum w_j U_j$ is fixed-parameter tractable when parameterized by $p_{\#} + w_{\#} + d_{\#}$. By Observation 1, from this immediately follows that $1 \mid r_j \mid \sum w_j U_j$ is also fixed-parameter tractable when parameterized by $p_{\#} + w_{\#} + r_{\#}$. To prove the second part of Theorem 9, present a reduction from $1 \mid r_j \mid \sum w_j U_j$ to MILP. Given an instance of $1 \mid r_j \mid \sum w_j U_j$, we create a number of instances of MILP where in each of them, the number of integer variables is upper-bounded by a function of $p_{\#}$, $w_{\#}$, and $d_{\#}$. We solve each instance using Theorem 2 and prove that the $1 \mid r_j \mid \sum w_j U_j$ instance admits a schedule where the weighted number of early jobs is at least W if and only if one of the generated instances admits a solution with objective value at least W . Furthermore, we can upper-bound the number of created MILP instances by a function of $p_{\#}$, $w_{\#}$, and $d_{\#}$.

Given an instance of $1 \mid r_j \mid \sum w_j U_j$, we say that two jobs j and j' have the same type if they have the same processing time, the same weight, and the same due date, that is, $p_j = p_{j'}$, $w_j = w_{j'}$, and $d_j = d_{j'}$. Let \mathcal{T} denote the set of all types. Note that we have $|\mathcal{T}| \leq p_{\#} \cdot w_{\#} \cdot d_{\#}$. For some $r \in \mathbb{N}_0$ and some $t \in \mathcal{T}$ we denote by $t(r)$ the set of jobs with type t whose release date is at least r . Further, we denote by p_t the processing time of jobs with type t . Let $d_1, \dots, d_{d_{\#}}$ be the sorted sequence of due dates. We denote with d_{ℓ} the ℓ^{th} due date and we use d_j to denote the due date of job j (same with release dates). To keep the notation concise we will use $d_0 = 0$ occasionally.

We fix some optimal schedule $\sigma : \{1, \dots, n\} \rightarrow \mathbb{N}$ for the instance so that we may guess some part of it by enumeration. If for a job j and a due date d_{ℓ} we have $\sigma(j) < d_{\ell} < \sigma(j) + p_j$, we will say that the job *overlaps* the due date. Notice that in any schedule, any due date is overlapped by at most one job, but a job may overlap multiple due dates.

We now want to enumerate all possible ways due dates can be overlapped by early jobs from some type $t \in \mathcal{T}$ in σ . That is, we consider all ways to partition $d_1, \dots, d_{d_{\#}}$ into subsequences of consecutive due dates. There are $2^{d_{\#}}$ such partitions. For each such subsequence S we consider all job types that might be scheduled overlapping all due dates in S . For the subsequences containing only a single due date, we also consider the case that no job overlaps that due date. This gives $p_{\#} \cdot w_{\#} \cdot d_{\#} + 1$ choices for each subsequence, of which there are $d_{\#}$. Thus we end up with at most $2^{d_{\#}} \cdot d_{\#}^{p_{\#} \cdot w_{\#} \cdot d_{\#} + 1}$ possible *overlap structures* to consider. By enumerating them it is now possible to assume that we know which overlap structure is present in σ .

We make a small simplification at this stage. Suppose we know that some sequence of due dates d_a, d_{a+1}, \dots, d_b is overlapped by the same job. Then σ schedules every job with one of these due dates such that it is either scheduled to end before d_a , or it is late. Therefore, we can decrease the due date of such jobs to be d_a without changing the optimality of σ . We may thus assume that every job overlaps at most one due date. So suppose that for each d_ℓ we are given the job type $T(d_\ell) = t$ overlapping that due date, or an assertion that no job overlaps d_ℓ , represented by $T(d_\ell) = \emptyset$. If the due date of jobs of type $T(d_\ell)$ is at most d_ℓ we can reject the arrangement immediately, so assume that the due date of jobs of type $T(d_\ell)$ is larger than d_ℓ . We can formulate a MILP that computes an optimal schedule under the constraint that this structure of overlaps is respected. It uses the following variables:

1. $x_t^\ell \in \mathbb{N}_0$ counts the number of jobs of type t to be scheduled between $d_{\ell-1}$ and d_ℓ .
2. $o_a^\ell, o_b^\ell \in \mathbb{N}_0$ are the portions of the job scheduled overlapping d_ℓ that is processed before and after d_ℓ , respectively.
3. $x_j \in [0, 1]$ indicates whether job j is scheduled. These variables are fractional, but we will be able to show that they can be rounded.

We could omit generating variables x_t^ℓ that are known to schedule jobs late, that is, those where jobs of type t have a due date that is earlier than d_ℓ . For simplicity, we keep these variables, but one can assume them to be set to 0.

The MILP needs the following sets of constraints. The first constraint sets handle the overlaps around due dates:

$$\forall \ell \in \{1, \dots, d_\# - 1\} : o_a^\ell + o_b^\ell = p_t, \text{ if } T(d_\ell) = t. \quad (6)$$

$$\forall \ell \in \{1, \dots, d_\# - 1\} : o_a^\ell + o_b^\ell = 0, \text{ if } T(d_\ell) = \emptyset. \quad (7)$$

$$\forall \ell \in \{1, \dots, d_\# - 1\} : d_\ell + o_b^\ell \leq d_{\ell+1}. \quad (8)$$

$$\forall \ell \in \{1, \dots, d_\# - 1\} : d_{\ell-1} + o_a^\ell \leq d_\ell. \quad (9)$$

The next set of constraints ensures that we do not try to schedule more jobs of a certain type than there are available:

$$\forall t \in \mathcal{T} : \sum_{j \text{ has type } t} x_j = \sum_{\ell \in \{1, \dots, d_\#\}} x_t^\ell + |\{d_\ell \mid T(d_\ell) = t\}|. \quad (10)$$

The next two sets of constraints, intuitively, ensure that we respect the release dates of the jobs, and that we do not schedule too many jobs between two consecutive due dates.

$$\forall \ell \in \{1, \dots, d_\#\}, \ell' \in \{1, \dots, r_\#\} \text{ with } r_{\ell'} \leq d_\ell$$

$$o_a^\ell + \sum_{t \in \mathcal{T}} p_t \cdot \left(\sum_{j \in t(r_{\ell'})} x_j - \sum_{\ell'' > \ell} x_t^{\ell''} - |\{d_{\ell''} \mid T(d_{\ell''}) = t, \ell'' \geq \ell\}| \right) \leq d_\ell - r_{\ell'}. \quad (11)$$

$$\forall \ell \in \{1, \dots, d_\#\} : o_a^\ell + o_b^{\ell-1} + \sum_{t \in \mathcal{T}} p_t \cdot x_t^\ell \leq d_\ell - d_{\ell-1}. \quad (12)$$

The objective function (to be maximized) of the MILP is simply

$$\sum_j w_j \cdot x_j. \quad (13)$$

We observe the following:

19:10 Single-Machine Scheduling to Minimize the Number of Tardy Jobs

► **Observation 10.** *The number of created MILP instances is in $O(2^{d_{\#}} \cdot d_{\#}^{p_{\#} \cdot w_{\#} \cdot d_{\#} + 1})$ and the number of integer variables in each instance is in $O(p_{\#} \cdot w_{\#} \cdot d_{\#}^2)$.*

Next, we show the correctness of the reduction.

► **Lemma 11 (*)**. *If the $1 \mid r_j \mid \sum w_j U_j$ instance admits a schedule where the weighted number of early jobs is W , then one of the created MILP instances admits a feasible solution that has objective value at least W .*

It remains to show that we can construct a schedule from a solution to the MILP. We will first show some auxiliary result for the case of a single due date.

► **Lemma 12.** *Consider any instance I of $1 \mid r_j \mid \sum w_j U_j$ with a common due date d . If for all release dates r_j we have*

$$\sum_{j' \in \mathcal{I} \mid r_{j'} \geq r_j} p_{j'} \leq d - r_j .$$

then we can schedule all jobs early.

Proof. The statement holds if the instance I contains a single job. Otherwise, we apply induction on the number of jobs. Let j^* be a job with maximum release date. We schedule that job at time $d - p_{j^*}$. This yields a new instance I' of $1 \mid r_j \mid \sum w_j U_j$ with one fewer job and common due date $d - p_{j^*}$. For any of the release dates r'_j of this new instance, it holds that

$$\sum_{j' \in I' \mid r_{j'} \geq r'_j} p_{j'} = \sum_{j' \in I' \mid r_{j'} \geq r'_j} p_{j'} - p_{j^*} \leq d - r'_j - p_{j^*} .$$

The lemma statement follows. ◀

► **Lemma 13.** *If one of the created MILP instances admits a solution that has objective value W , then the $1 \mid r_j \mid \sum w_j U_j$ instance admits a schedule where the weighted number of early jobs is at least W .*

Proof. Assume we are given a feasible solution with objective value W to one of the MILP instances. We will begin by rounding the x_j such that they are integral. Suppose there is some job j with $x_j \in (0, 1)$. Due to constraint (10) there exists some other job j' with the same type as j and with $x_{j'} \in (0, 1)$. Assume, without loss of generality, that j has an earlier release date than j' . Then we claim that setting $x_j := \min\{1, x_j + x_{j'}\}$ and $x_{j'} := \max\{0, x_{j'} + x_j - 1\}$ maintains feasibility. Note that $x_j + x_{j'}$ does not change, $x_{j'}$ decreases, and in any constraint where x_j occurs, $x_{j'}$ occurs also because j has the earlier release date and they have the same type. Therefore, the left-hand sides of constraint (10) are unchanged, and those of constraint (11) do not increase, maintaining feasibility. Since j and j' have the same type, the objective value is unchanged. By repeating this rounding operation, we can increase the number of integral x_j until all of them are integral. The same argument can also be used to ensure that no job j has $x_j = 1$ if there is another job j' with the same type and an earlier release date having $x_{j'} = 0$.

We now show how to schedule every early job j with $x_j = 1$. Note that this implies that the weighted number of early jobs is at least W . First, let $J_t^1, \dots, J_t^{r_{\#}}$ be the list of jobs with type t sorted by their release dates, and omitting all jobs with $x_j = 0$. It will suffice to prove that we can schedule the last $x_t^{d_{\#}}$ jobs from each list into the interval $\mathcal{I} := [d_{d_{\#}-1} + o_b^{d_{\#}-1}, d_{d_{\#}}]$, and that we can then schedule a job of type $T(d_{d_{\#}-1})$ into $[d_{d_{\#}-1} - o_a^{d_{\#}-1}, d_{d_{\#}-1} + o_b^{d_{\#}-1}]$.

Notice that constraint (11) simplifies for $d_{d\#}$ to be $\sum_{t \in \mathcal{T}} \sum_{j \in t(r_{\ell'})} p_t \cdot x_j \leq d_{d\#} - r_{\ell'}$. Further, for the purposes of scheduling into \mathcal{I} , we may consider the $x_t^{d\#}$ jobs from each type t with the latest release dates to have release date at least $d_{d\#-1} + o_b^{d\#-1}$, which transforms constraint (12) into $\sum_{t \in \mathcal{T}} \sum_{j \in t(d_{d\#-1} + o_b^{d\#-1})} p_t \cdot x_j \leq d_{d\#} - d_{d\#-1} - o_b^{d\#-1}$. From Lemma 12 we see that we can indeed schedule all the required jobs into \mathcal{I} .

Remove all the jobs we just scheduled from the instance, and update the MILP solution by setting to 0 the $x_t^{d\#}$, as well as all x_j for the scheduled jobs. Now we need to schedule the job j^* with the latest due date from the jobs of type $T(d_{d\#-1})$ into $[d_{d\#-1} - o_a^{d\#-1}, d_{d\#-1} + o_b^{d\#-1}]$. By constraints (6)-(9) we see that the interval has the correct length for the job, and that the job will not be late. We only need to ensure that it is not scheduled before its release date r_{j^*} . We use constraint (11) to observe

$$\begin{aligned} & o_a^{d\#-1} + \sum_{t \in \mathcal{T}} p_t \cdot \left(\sum_{j \in t(r_{j^*})} x_j - \sum_{\ell'' > d_{d\#-1}} x_t^{\ell''} - |\{d_{\ell''} \mid T(d_{\ell''}) = t, \ell'' \geq d_{d\#} - 1\}| \right) \leq d_{d\#-1} - r_{j^*} \\ & \implies r_{j^*} - p_{j^*} + \sum_{t \in \mathcal{T}} p_t \cdot \sum_{j \in t(r_{j^*})} x_j \leq d_{d\#-1} - o_a^{d\#-1} \\ & \implies r_{j^*} \leq d_{d\#-1} - o_a^{d\#-1}. \end{aligned}$$

We see that j^* can be scheduled as desired. Now we update the MILP solution again by setting $d_{d\#-1}$ to $d_{d\#-1} - o_a^{d\#-1}$, as well as x_{j^*} , $o_b^{d\#-1}$, and $o_a^{d\#-1}$ to 0. We also update $T(d_{d\#-1}) := \emptyset$. This yields a feasible solution to the MILP of the residual instance where we discard the constraints for the final due date. We can iterate this until all jobs j with $x_j = 1$ have been scheduled. We schedule the remaining jobs in an arbitrary but feasible way. Since the objective value of the solution for the MILP is W , we have that the weighted number of early jobs is also at least W . ◀

Now we have all the pieces available that are needed to prove Theorem 9.

Proof of Theorem 9. Between Lemma 11 and Lemma 13 we see that the MILP is equivalent to the original scheduling problem if the correct overlap structure is chosen. By Observation 10 the number of MILPs we need to solve and the number of integer variables in each MILP depends only on $p_{\#} + w_{\#} + d_{\#}$, and thus Theorem 9 follows from Theorem 2 and Observation 1. ◀

6 Unary 1 | r_j | $\sum w_j U_j$ parameterized by $r_{\#}$

Recall that $1 | r_j | \sum w_j U_j$ is *weakly* NP-hard in the case where there is only one due date and one release date. Thus even for instances of $1 | r_j | \sum w_j U_j$ with constant $r_{\#}$ or $d_{\#}$, we cannot expect a polynomial-time algorithm solving them in general. However, we show in the following that such instances can be solved in *pseudo-polynomial time*², using dynamic programming, thus generalizing the folklore algorithm for KNAPSACK. Formally, we prove the following:

² A problem can be solved in *pseudo-polynomial* time if it can be solved in polynomial time when all numeric values are encoded unarily.

19:12 Single-Machine Scheduling to Minimize the Number of Tardy Jobs

► **Theorem 14.** *The problem $1 \mid r_j \mid \sum w_j U_j$ is in XP when parameterized by $r_{\#}$ or $d_{\#}$ if all numbers are encoded in unary.*

By Observation 1, we can focus on $r_{\#}$ as a parameter, and the result for $d_{\#}$ as a parameter follows. We begin by ordering the release dates and denote with r_{ℓ} and r_k the ℓ^{th} and k^{th} release date, respectively, and we use r_i and r_j to denote the release date of jobs i and j , respectively (same with due dates). We also use \leq_d to denote a fixed order of the jobs such that their due dates are non-descending. We encode this directly into the indexing of the jobs, that is, $i \leq_d j$ if and only if $i \leq j$.

We can now assume that there exists some optimal schedule such that at every release date r_{ℓ} there is a job scheduled to start exactly at r_{ℓ} . This can be ensured by enumerating the possible overlap structures of the optimal schedule with respect to the release dates, as in Section 5. To do this, we need to guess for each release date r_{ℓ} if there is a job scheduled there and what the completion time of that job is. We know the completion time to be in $[r_{\ell} + 1, r_{\ell} + p_{\max}]$, so in reality we will need to solve $(p_{\max})^{r_{\#}}$ dynamic programs and return the best solution found. This is only a pseudo-polynomial time overhead, so we will suppress it in the following.

Notice that between two consecutive release dates, we can now assume the scheduled early jobs to be ordered according to \leq_d , that is, by the earliest due date (all tardy jobs are scheduled later on in a feasible but arbitrary way). If they are not ordered as such two adjacent out-of-order jobs can be swapped while maintaining feasibility.

This structural simplification allows us to write a dynamic program with the following recursive definition of the dynamic programming table:

$$T[j, t_1, \dots, t_{r_{\#}}] = \max_{\substack{\ell \text{ with } r_{\ell} \geq r_j \\ \text{and } r_{\ell} + t_{\ell} \leq d_j}} \{T[j-1, t_1, \dots, t_{r_{\#}}], T[j-1, t_1, \dots, t_{\ell} - p_j, \dots, t_{r_{\#}}] + w_j\}.$$

The base cases for the recursion are:

- $T[0, 0, \dots, 0] = 0$,
- $T[0, t_1, \dots, t_{r_{\#}}] = -\infty$ if any of the t_{ℓ} are not zero,
- $T[j, \cdot, t_{\ell}, \cdot] = -\infty$ if $t_{\ell} > r_{\ell+1} - r_{\ell}$ or $t_{\ell} < 0$ for some ℓ .

The entry $T[j, t_1, \dots, t_{r_{\#}}]$ intuitively represents the most valuable schedule attainable using only the first j jobs according to \leq_d , and with a total scheduled job time of t_{ℓ} starting at r_{ℓ} .

We will first notice that the DP-table T has at most $n \cdot (n \cdot p_{\max})^{r_{\#}}$ finite entries. Furthermore, each of those entries can be computed in time $O(r_{\#})$ if we process them in increasing order of the first index. It remains to show that there exists a schedule attaining the value $\max_{t_1, \dots, t_{r_{\#}}} (T[|J|, t_1, \dots, t_{r_{\#}}])$, as well as that there is some cell $T[|J|, t_1, \dots, t_{r_{\#}}]$ that has value at least as high as that of an optimal schedule.

► **Lemma 15** (\star). *If the $1 \mid r_j \mid \sum w_j U_j$ instance admits a schedule where the weighted number of early jobs is W , then there exists some entry of the dynamic programming table T with value at least W .*

► **Lemma 16** (\star). *Let $T[i, t_1, \dots, t_{r_{\#}}]$ be a cell of the dynamic program with finite value W . Then there is a schedule where at most the first i jobs are early, that attains a weighted number of early jobs W , and which schedules a total processing time of t_{ℓ} immediately after release date r_{ℓ} .*

We may now conclude the following:

► **Corollary 17.** Let $W = \max_{t_1, \dots, t_{r_\#}} \{T[n, t_1, \dots, t_{r_\#}]\}$. Then W is the maximum weighted number of jobs that can be scheduled early in the $1 \mid r_j \mid \sum w_j U_j$ instance.

Proof. By Lemma 15 we have that the maximum weighted number of jobs that can be scheduled early in the $1 \mid r_j \mid \sum w_j U_j$ instance is at least W . However, let $T[n, t_1, \dots, t_{r_\#}]$ be an entry with value W . Then Lemma 16 guarantees that there exists a schedule for the $1 \mid r_j \mid \sum w_j U_j$ instance such that the weighted number of early jobs is W . ◀

Now we have all the pieces to prove a running time upper bounds for our dynamic-programming algorithm for $1 \mid r_j \mid \sum w_j U_j$.

► **Lemma 18.** The problem $1 \mid r_j \mid \sum w_j U_j$ can be solved in time $O(p_{\max}^{r_\#} \cdot n \cdot (n \cdot p_{\max})^{r_\#} \cdot r_\#)$.

Proof. We first need to guess the correct overlap structure between jobs and release dates of which there are at most $p_{\max}^{r_\#}$ many. For each of these overlap structures, we then compute the dynamic programming in time $O(n \cdot (n \cdot p_{\max})^{r_\#} \cdot r_\#)$. We finally return the best objective value found in any of the dynamic programming tables. By Corollary 17, this is the maximum weighted number of jobs that can be scheduled early in the $1 \mid r_j \mid \sum w_j U_j$ instance. ◀

Now we have all the pieces to prove Theorem 14.

Proof of Theorem 14. Theorem 14 follows directly from Lemma 18 and Observation 1. ◀

Notice that the proof of Lemma 16 also gives a backtracking procedure which allows us to compute an optimal schedule in time $O(n \cdot r_\#)$ for a given filled in dynamic programming table with the maximum entry already computed.

7 Conclusion

In this work, we give a comprehensive overview of the parameterized complexity of $1 \mid r_j \mid \sum w_j U_j$ for the parameters number $p_\#$ of processing times, number $w_\#$ of weights, number $r_\#$ of release dates, and number $d_\#$ of due dates. We leave several questions for future research, in particular:

- Is $1 \mid r_j \mid \sum w_j U_j$ in XP when parameterized by $p_\#$?
- Is $1 \mid r_j \mid \sum w_j U_j$ fixed-parameter tractable when parameterized by $p_\# + w_\#$?
- Is $1 \mid r_j \mid \sum w_j U_j$ fixed-parameter tractable when parameterized by $p_\#$ and all numbers are encoded unarily?

We remark that a technical report [4] claims (strong) NP-hardness even for $p_\# = 2$ and $w_\# = 1$. If that result holds, then it would also settle the parameterized complexity for the parameter combination $p_\# + w_\#$, and for $p_\#$ even when all processing times and weights are encoded in unary.



References

- 1 Esther M. Arkin and Ellen B. Silverberg. Scheduling jobs with fixed start and end times. *Discrete Appl. Math.*, 18(1):1–8, 1987. doi:10.1016/0166-218X(87)90037-0.
- 2 Philippe Baptiste, Peter Brucker, Sigrid Knust, and Vadim G Timkovsky. Ten notes on equal-processing-time scheduling: at the frontiers of solvability in polynomial time. *Quarterly J. Belgian, French and Ital. Oper. Res. Soc.*, 2(2):111–127, 2004. doi:10.1007/s10288-003-0024-4.
- 3 Karl Bringmann, Nick Fischer, Danny Hermelin, Dvir Shabtay, and Philip Wellnitz. Faster minimization of tardy processing time on a single machine. *Algorithmica*, 84(5):1341–1356, 2022. doi:10.1007/S00453-022-00928-W.



- 4 Jan Elffers and Mathijs de Weerd. Scheduling with two non-unit task lengths is NP-complete. Technical report, 2014. doi:10.48550/arXiv.1412.3095.
- 5 Michael R. Fellows, Serge Gaspers, and Frances A. Rosamond. Parameterizing by the number of numbers. *Theory Comput. Syst.*, 50(4):675–693, 2012. doi:10.1007/S00224-011-9367-Y.
- 6 Nick Fischer and Leo Wennmann. Minimizing tardy processing time on a single machine in near-linear time. Technical report, 2024. doi:10.48550/arXiv.2402.13357.
- 7 M. R. Garey, D. S. Johnson, B. B. Simons, and R. E. Tarjan. Scheduling unit-time tasks with arbitrary release times and deadlines. *SIAM J. Comput.*, 10(2):256–269, 1981. doi:10.1137/0210018.
- 8 Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math.*, 17(2):416–429, 1969. doi:10.1137/0117039.
- 9 Klaus Heeger and Danny Hermelin. Minimizing the weighted number of tardy jobs is $W[1]$ -hard. In *Proc. ESA 2024*, volume 308 of *Leibniz Int. Proc. Informatics*, pages 68:1–68:14, 2024. doi:10.4230/LIPICS.ESA.2024.68.
- 10 Klaus Heeger, Danny Hermelin, George B. Mertzios, Hendrik Molter, Rolf Niedermeier, and Dvir Shabtay. Equitable scheduling on a single machine. *J. Sched.*, 26(2):209–225, 2023. doi:10.1007/S10951-022-00754-6.
- 11 Klaus Heeger and Hendrik Molter. Minimizing the number of tardy jobs with uniform processing times on parallel machines. Technical report, 2024. doi:10.48550/arXiv.2404.14208.
- 12 Danny Hermelin, Yuval Itzhaki, Hendrik Molter, and Dvir Shabtay. On the parameterized complexity of interval scheduling with eligible machine sets. *J. Comput. Syst. Sci.*, page 103533, 2024. doi:10.1016/J.JCSS.2024.103533.
- 13 Danny Hermelin, Shlomo Karhi, Michael L. Pinedo, and Dvir Shabtay. New algorithms for minimizing the weighted number of tardy jobs on a single machine. *Ann. Oper. Res.*, 298(1):271–287, 2021. doi:10.1007/S10479-018-2852-9.
- 14 Danny Hermelin, Hendrik Molter, and Dvir Shabtay. Minimizing the weighted number of tardy jobs via $(\max, +)$ -convolutions. *INFORMS J. Comput.*, 36:836–848, 2024. doi:10.1287/IJOC.2022.0307.
- 15 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.
- 16 Matthias Kaul, Matthias Mnich, and Hendrik Molter. Single-machine scheduling to minimize the number of tardy jobs with release dates. Technical report, 2024. doi:10.48550/arXiv.2408.12967.
- 17 Kim-Manuel Klein, Adam Polak, and Lars Rohwedder. On minimizing tardy processing time, max-min skewed convolution, and triangular structured ILPs. In *Proc. SODA 2023*, pages 2947–2960, 2023. doi:10.1137/1.9781611977554.CH112.
- 18 Sven O. Krumke, Clemens Thielen, and Stephan Westphal. Interval scheduling on related machines. *Comput. Oper. Res.*, 38(12):1836–1844, 2011. doi:10.1016/J.COR.2011.03.001.
- 19 Eugene L. Lawler and James M. Moore. A functional equation and its application to resource allocation and sequencing problems. *Mgmt. Sci.*, 16(1):77–84, 1969. doi:10.1287/mnsc.16.1.77.
- 20 Jan K. Lenstra, A.H.G. Rinnooy Kan, and Peter Brucker. Complexity of machine scheduling problems. *Ann. Discrete Math.*, 1:343–362, 1977. doi:10.1016/S0167-5060(08)70743-X.
- 21 Hendrik W Lenstra Jr. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983. doi:10.1287/MOOR.8.4.538.
- 22 Matthias Mnich and René van Bevern. Parameterized complexity of machine scheduling: 15 open problems. *Comput. Oper. Res.*, 100:254–261, 2018. doi:10.1016/J.COR.2018.07.020.
- 23 Matthias Mnich and Andreas Wiese. Scheduling and fixed-parameter tractability. *Math. Prog.*, 154(1):533–562, 2015. doi:10.1007/S10107-014-0830-9.
- 24 James M. Moore. An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Mgmt. Sci.*, 15:102–109, 1968. doi:10.1287/mnsc.15.1.102.
- 25 Michael Pinedo. *Scheduling: Theory, Algorithms, and Systems, 5th Edition*. Springer, 2016.

- 26 Baruch Schieber and Pranav Sitaraman. Quick minimization of tardy processing time on a single machine. In *Proc. WADS 2023*, volume 14079 of *Lecture Notes Comput. Sci.*, pages 637–643, 2023. doi:10.1007/978-3-031-38906-1_42.
- 27 Shao Chin Sung and Milan Vlach. Maximizing weighted number of just-in-time jobs on unrelated parallel machines. *J. Sched.*, 8(5):453–460, 2005. doi:10.1007/S10951-005-2863-7.

Quasi-Linear Distance Query Reconstruction for Graphs of Bounded Treelength

Paul Bastide  

LaBRI – Université de Bordeaux, France
TU Delft, The Netherlands

Carla Groenland  

TU Delft, The Netherlands

Abstract

In distance query reconstruction, we wish to reconstruct the edge set of a hidden graph by asking as few distance queries as possible to an oracle. Given two vertices u and v , the oracle returns the shortest path distance between u and v in the graph.

The *length* of a tree decomposition is the maximum distance between two vertices contained in the same bag. The *treelength* of a graph is defined as the minimum length of a tree decomposition of this graph. We present an algorithm to reconstruct an n -vertex connected graph G parameterized by maximum degree Δ and treelength k in $O_{k,\Delta}(n \log^2 n)$ queries (in expectation). This is the first algorithm to achieve quasi-linear complexity for this class of graphs. The proof goes through a new lemma that could give independent insight on graphs of bounded treelength.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Distance Reconstruction, Randomized Algorithm, Treelength

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.20

1 Introduction

There has been extensive study on identifying the structure of decentralized networks [2, 11, 15, 12, 1]. These networks are composed of vertices (representing servers or computers) and edges (representing direct interconnections). To trace the path through these networks from one actor to another, tools like `traceroute` (also known as `tracert`) were developed. If the entire route cannot be inferred (e.g. due to privacy concerns), a ping-pong protocol can be employed in which one node sends a dummy message to the second node, which then immediately responds with a dummy message back to the first node. This process aims to infer the distance between the nodes by measuring the time elapsed between the sending of the first message and the receipt of the second.

A mathematical model for this called the *distance query model* was introduced [2]. In this model, only the vertex set V of a hidden graph $G = (V, E)$ is known, and the goal is to reconstruct the edge set E through distance queries to an oracle. For any pair of vertices $(u, v) \in V^2$, the oracle provides the shortest path distance between u and v in G . The algorithm can be adaptive and base its next query on the responses to previous queries.

For a graph class \mathcal{G} of connected graphs, an algorithm is said to reconstruct the graphs in the class if, for every graph $G \in \mathcal{G}$, the distance profile obtained is unique to G within \mathcal{G} . We then say the graph has been reconstructed. The *query complexity* refers to the maximum number of queries the algorithm executes on an input graph from \mathcal{G} . For a randomised algorithm, the query complexity is determined by the expected number of queries, accounting for the algorithm's randomness. Such a randomised algorithm could also be seen as a probability distribution over decision trees.



© Paul Bastide and Carla Groenland;

licensed under Creative Commons License CC-BY 4.0

19th International Symposium on Parameterized and Exact Computation (IPEC 2024).

Editors: Édouard Bonnet and Paweł Rzażewski; Article No. 20; pp. 20:1–20:11

Leibniz International Proceedings in Informatics



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

Note that querying the oracle for the distance between every pair of vertices in G would reconstruct the edge set as $E = \{\{u, v\} \mid d(u, v) = 1\}$. This approach leads to a trivial upper bound of $|V|^2$ on the query complexity. Unfortunately, $\Omega(|V|^2)$ queries may be required to, for example, distinguish between a clique K_n and K_n minus an edge $\{u, v\}$. If the maximum degree is unbounded, this issue persists even in sparse graphs like trees: it can take $\Omega(n^2)$ queries to distinguish n -vertex trees (see also [13]). Therefore, as was also done in earlier work, we will restrict ourselves to connected n -vertex graphs with maximum degree Δ .

1.1 Previous work

Kannan, Mathieu and Zhou [9, 11] were the first to give a non-trivial upper bound for all graphs of bounded maximum degree, designing a randomised algorithm using $\tilde{O}_\Delta(n^{3/2})$ queries in expectation for n -vertex graphs of maximum degree Δ . Here $\tilde{O}(f(n))$ stands for $O(f(n) \text{polylog}(n))$ and the Δ subscript denotes that Δ is considered a parameter and only influences the multiplicative constant in front of $f(n)$, (e.g here we mean $g(\Delta)n^{3/2} \text{polylog } n$ for some function $g : \mathbb{N} \mapsto \mathbb{R}$.) This is still the best known upper bound in the general case, while the best lower bound is $\Omega(\Delta n \log_\Delta n)$ [1]. Researchers spent effort investigating $\tilde{O}_\Delta(n)$ algorithms for restricted classes of graphs. Kannan, Mathieu and Zhou [9, 11] proved that there exists an $O_\Delta(n \log^3 n)$ randomised algorithm for chordal graphs (graphs without induced cycle of length at least 4). Since then, their algorithm for chordal graphs has been improved by Rong, Li, Yang, and Wang [15] to $O_\Delta(n \log^2 n)$, who also extended the class to 4-chordal graphs (graphs without induced cycle of length at least 5). Recent works introduced new techniques to design deterministic reconstruction algorithms [1]. They developed a quasi-linear algorithm for bounded maximum degree k -chordal graphs (without induced cycle of length at least $k + 1$ and maximum degree Δ) using $O_{\Delta,k}(n \log n)$ queries. Their results can be interpreted as a quasi-linear algorithm parameterized by maximum degree and chordality. In this paper, we are the first to use a parameterized approach to extend on the techniques of Kannan, Mathieu and Zhou [9, 11], obtaining an algorithm with quasi-linear query complexity parametrized by even more general parameters.

1.2 Treelength

A graph G has *treelength* at most ℓ if it admits a tree decomposition such that $d_G(u, v) \leq \ell$ whenever $u, v \in V(G)$ share of a bag (see Section 2 for formal definition). We emphasize that the bags are allowed to induce disconnected subgraphs, and that the “bounded diameter” constraint is measured within the entire graph. Graphs of treelength 1 are exactly chordal graphs and it was proved in [10] that k -chordal graphs have treelength at most k . For $k > 1$, the class of graphs of treelength at most k covers a larger class of graphs than the class of k -chordal graphs.

Graphs of bounded treelength avoid long geodesic cycles (i.e. cycles C for which $d_C(x, y) = d_G(x, y)$ for all $x, y \in C$) and in fact bounded treelength is equivalent to avoiding long “loaded geodesic cycles” or being “boundedly quasi-isometric to a tree” (see [4] for formal statements). When a graph has bounded treewidth (defined in Section 2), then the length of the longest geodesic cycle is bounded if and only if the *connected treewidth* is bounded [5]. In a tree decomposition of connected treewidth at most k , bags induce connected subgraphs of size at most $k + 1$, which in particular means that graph distance between vertices sharing a bag is at most k . So for graphs of bounded treewidth, excluding long geodesic cycles is in fact equivalent to bounding the treelength of the graph.

Treelength has been extensively studied from an algorithmic standpoint, particularly for problems related to shortest path distances. For example, there exist efficient routing schemes for graphs with bounded treelength [7, 10] and an FPT algorithm for computing the metric dimension of a graph parameterised by its treelength [3]. Although deciding the treelength of a given graph is NP-complete, it can still be approximated efficiently [7, 6].

1.3 Our contribution

Building on methods used by Kannan, Mathieu, and Zhou [11, 9] to reconstruct chordal graphs, we prove the following result.

► **Theorem 1.** *There is a randomised algorithm that reconstructs an n -vertex graph of maximum degree at most Δ and treelength at most k using $O_{\Delta,k}(n \log^2 n)$ distance queries in expectation.*

We now first describe the technique used by Kannan, Mathieu and Zhou [11, 9] for chordal graphs and then discuss our extension. In their approach, they design a clever subroutine to compute a small balanced separator S of the graph G using $\tilde{O}_{\Delta}(n)$ queries. With the knowledge of this separator, it is possible to compute the partition in connected component of $G \setminus S$. By using this subroutine recursively, they are able to decompose the graph into smaller and smaller components until a brute-force search already yields a $\tilde{O}_{\Delta}(n)$ queries algorithm. They exploit the strong structure of chordal graphs in two ways in this algorithm. First, to compute a small separator S . They start by only finding a single vertex that lies on many shortest paths. They then use a specific tree decomposition of chordal graphs, where all bags are cliques, to argue that the neighbourhood of this vertex is a good separator. Second, they show that for any connected component C of $G \setminus S$ the distance between vertices in C are the same in $G[C \cup S]^1$ and in G . This property allows to apply their subroutine recursively, as we can now simulate a distance oracle in $G[C \cap S]$ by just using the one we have on G .

Theorem 1 shows that we can push the boundaries of such an approach, and proves that a weaker condition on the tree decomposition is already sufficient. We weaken the “bags are cliques” condition, satisfied by chordal graphs, to the weaker condition “bags have bounded diameter”. The bags are not required to be connected: the diameter is measured in terms of the distance between the vertices in the entire graph.

We provide a brief explanation of our method and highlight the new challenges compared to the approaches in [11] and [9]. We also start by finding a vertex v that lies on many shortest paths (with high probability), although we give a new approach for doing so. In fact, our overall algorithm is more efficient than that of [11, 9] by a $(\log n)$ -factor, and this is the place where we gain this improvement. We then show that for such a vertex v , the set $S = N^{\leq 3k/2}[v]$ of vertices at distance at most $3k/2$ is a good separator, for k the treelength of the input graph. We compute the components of $G \setminus S$ to check that indeed we found a good separator and then recursively reconstruct the components until we reach a sufficiently small vertex set on which a brute-force approach can be applied. It is key to our recursive approach, and requires non-trivial proofs, that we can add a small boundary set and still preserve all the relevant distances for a component. This problem is easily avoided in [11, 9] where separators are cliques, but is more delicate to handle in our case. For this, we amongst others obtain a structural property of graphs with bounded treelength. This property is stated in the following lemma, which may be of independent interest.

¹ Given a graph G and a set of vertices $S \subseteq V(G)$, we use the notation $G[S]$ to denote the graphs induces by G on the vertex set S .

► **Lemma 2.** *Let G be a graph of treelength at most $k \geq 1$ and $A \subseteq V(G)$. If $G[A]$ is connected then every shortest path in G between two vertices $a, b \in A$ is contained in $N^{\leq 3k/2}[A]$.*

1.4 Roadmap

In Section 2, we set up our notation and give the relevant definitions. In Section 3, we give our algorithm to reconstruct bounded treelength graph with a proof of correctness and complexity. In Section 4 we conclude with some open problems.

2 Preliminaries

In this paper, all graphs are simple, undirected and connected except when stated otherwise. All logarithms in this paper are base 2, unless mentioned otherwise. For $a \leq b$ two integers, let $[a, b]$ denote the set of all integers x satisfying $a \leq x \leq b$. We short-cut $[a] = [1, a]$.

For a graph G and two vertices $a, b \in V(G)$, we denote by $d_G(a, b)$ the length of a shortest path between a and b . For $G = (V, E)$, $A \subseteq V$ and $i \in \mathbb{N}$, we denote by $N_G^{\leq i}[A] = \{v \in V \mid \exists a \in A, d_G(v, a) \leq i\}$. We may omit the superscript when $i = 1$. We write $N_G(A) = N_G[A] \setminus A$ and use the shortcuts $N_G[u], N_G(u)$ for $N_G[\{u\}], N_G(\{u\})$ when u is a single vertex. We may omit the subscript when the graph is clear from the context.

2.1 Distance queries

We denote by $\text{QUERY}_G(u, v)$ the call to an oracle that answers $d_G(u, v)$, the distance between u and v in a graph G . For A, B two sets of vertices, we denote by $\text{QUERY}_G(A, B)$ the $|A| \cdot |B|$ calls to an oracle, answering the list of distances $d_G(a, b)$ for all $a \in A$ and all $b \in B$. We may abuse notation and write $\text{QUERY}_G(u, A)$ for $\text{QUERY}_G(\{u\}, A)$ and may omit G when the graph is clear from the context.

For a graph class \mathcal{G} of connected graphs, we say an algorithm reconstructs the graphs in the class if for every graph $G \in \mathcal{G}$ the distance profile obtained from the queries is not compatible with any other graph from \mathcal{G} . The *query complexity* is the maximum number of queries that the algorithm takes on an input graph from \mathcal{G} , where the queries are adaptive. For a randomised algorithm, the query complexity is given by the expected number of queries (with respect to the randomness in the algorithm).

2.2 Tree decomposition

A *tree decomposition* of a graph G is a tuple $(T, (B_t)_{t \in V(T)})$ where T is a tree and B_t is a subset of $V(G)$ for every $t \in V(T)$, for which the following conditions hold.

- For every $v \in V(G)$, the set $\{t \in V(T) \mid v \in B_t\}$ is non-empty and induces a subtree of T .
- For every $uv \in E(G)$, there exists a $t \in V(T)$ such that $\{u, v\} \subseteq B_t$.

This notion was introduced by [14].

2.3 Treelength

The *treelength* of a graph G (denoted $\text{tl}(G)$) is the minimal integer k for which there exists a tree decomposition $(T, (B_t)_{t \in V(T)})$ of G such that $d(u, v) \leq k$ for every pair of vertices u, v that share a bag (i.e. $u, v \in B_t$ for some $t \in V(T)$). We refer the reader to [7] for a detailed overview of the class of bounded treelength graphs.

2.4 Balanced separators

For $\beta \in (0, 1)$, a β -balanced separator of a graph $G = (V, E)$ for a vertex set $A \subseteq V$ is a set S of vertices such that the connected components of $G[A \setminus S]$ are of size at most $\beta|A|$.

One nice property of tree decompositions is that they yield $\frac{1}{2}$ -balanced separators.

► **Lemma 3** ([14]). *Let G be a graph, $A \subseteq V(G)$ and $(T, (B_t)_{t \in V(T)})$ a tree decomposition of G . Then there exists $t \in V(T)$ such that B_t is a $\frac{1}{2}$ -balanced separator of A in G .*

3 Randomised algorithm for bounded treelength

We give the complete proof of Theorem 1 in this section.

► **Theorem 1.** *There is a randomised algorithm that reconstructs an n -vertex graph of maximum degree at most Δ and treelength at most k using $O_{\Delta, k}(n \log^2 n)$ distance queries in expectation.*

Given a tree decomposition $(T, (B_t)_{t \in V(T)})$ of a graph G and a set X of vertices of G , we denote by T_X the subtree of T induced by the set of vertices $t \in V(T)$ such that B_t contains at least one vertex of X . Given $v \in V(G)$, we may abuse notation and use T_v as the subtree $T_{\{v\}}$. We first prove the following useful property of graphs of bounded treelength.

► **Lemma 2.** *Let G be a graph of treelength at most $k \geq 1$ and $A \subseteq V(G)$. If $G[A]$ is connected then every shortest path in G between two vertices $a, b \in A$ is contained in $N^{\leq 3k/2}[A]$.*

Proof. Consider a tree decomposition $(T, (B_t)_{t \in V(T)})$ of G such that any two vertices u, v in the same bag satisfy $d(u, v) \leq k$. If two vertices $a, b \in A$ share a bag, then $d(a, b) \leq k$ and the claim holds for this pair.

Otherwise, T_a and T_b are disjoint subtrees of T and we can consider the unique path P in T between T_a and T_b , with internal nodes taken from $V(T) \setminus V(T_a) \cup V(T_b)$. We also consider a shortest path $Q := \{q_1, q_2, \dots, q_m\}$ between a and b in G with $q_1 = a, q_m = b$ and $q_i q_{i+1} \in E(G)$ for all $i < m$. Since A is supposed connected, T_A is well-defined and is a subtree of T . Moreover T_A contains both T_a and T_b . Because T_A is a tree, it must then contain P as the unique path between T_a and T_b . Suppose now, towards a contradiction, that there is some vertex $z \in Q$ such that $z \notin N^{\leq 3k/2}[A]$. Note that T_z can not have common vertices with P because we assumed $d(z, A) > k$ using the previous remark and the fact that vertices that share a bag are at distance at most k . We can then consider the vertex $t \in P$ such that $\{t\}$ separates $P \setminus \{t\}$ from T_z in T . The shortest path Q must go through B_t twice: once to go from a to z and once to go from z to b .

Let $i < \ell < j$ be given such that $q_i, q_j \in B_t$ and $q_\ell = z$. Since Q is a shortest path in G , $d(q_i, z) + d(z, q_j) = d(q_i, q_j)$. Moreover, $d(q_i, q_j) \leq k$ because q_i and q_j share a bag. By the pigeonhole principle, we deduce that either $d(p_i, z) \leq k/2$ or $d(p_j, z) \leq k/2$. Suppose that $d(p_i, z) \leq k/2$. Remember that $t \in P$ thus B_t contains an element of A as $G[A]$ is connected. It follows that $d(p_i, A) \leq k$ thus $d(z, A) \leq d(z, p_i) + d(p_i, A) \leq 3k/2$, which is a contradiction. The other case follows by a similar argument. ◀

We now sketch the proof of Theorem 1. The skeleton of the proof is inspired by [9]: we find a balanced separator S , compute the partition of $G \setminus S$ into connected components, and reconstruct each component recursively. In order to find this separator, we use a notion of *betweenness* that roughly models the number of shortest paths a vertex is on.

20:6 Quasi-Linear Distance Query Reconstruction for Graphs of Bounded Treelength

We prove four claims. The first one ensures that in graphs of bounded treelength, the *betweenness* is always at least a constant. Then, the next three claims are building on each other to form an algorithm that computes the partition of $G \setminus S$ into connected components of roughly the same size.

- Claim 5 is a randomised procedure for finding a vertex z with high betweenness (using few queries and with constant success probability).
- Claim 6 shows $S = N^{\leq 3k/2}[z]$ is a good balanced separator if z has high betweenness.
- Claim 7 computes the partition of $G \setminus S$ into connected components. Note that, once you computed the partition, you can check if the preceding algorithms have been successful. If not, we can call again Claim 6 until we are successful, yielding a correct output with a small number of queries in expectation.

Proof of Theorem 1. Let G be a connected n -vertex graph of maximum degree at most Δ and let $(T, (B_t)_{t \in V(T)})$ be a tree decomposition of G such that $d(u, v) \leq k$ for all $u, v \in V(G)$ that share a bag in T .

We initialize $A = V(G)$, $n_A = |A|$ and $R^i = \emptyset$ for $i \in [1, 3k]$. For any $j \in \mathbb{R}^+$ we abbreviate $R^{\leq j} = \cup_{i \leq j} R^i$. Lastly, let $r = |R^{\leq 3k}|$. We will maintain throughout the following properties:

1. $G[A]$ is a connected induced subgraph of G .
 2. R^i consists of the vertices in G that are at distance exactly i from A .
 3. Both A and R^i for all i are known by the algorithm.
- In particular, we know which vertices are in sets such as $R^{\leq 3k/2} = N^{\leq 3k/2}[A]$ and by Lemma 2 we also obtain the following crucial property.

4. For $a, b \in A$, any shortest path between a and b only uses vertices from $A \cup R^{\leq 3k/2}$.
- The main idea of the algorithm is to find a balanced separator S and compute the partition of $G[A \setminus S]$ into connected components, then call the algorithm recursively on each components. As soon as n_A has become sufficiently small, we will reconstruct $G[A]$ by “brute-force queries”.

In order to find the separator S , we use the following notion. For G a graph, a subset $A \subseteq V(G)$ and a vertex $v \in V(G)$, the betweenness $p_v^G(A)$ is the fraction of pairs of vertices $\{a, b\} \subseteq A$ such that v is on some shortest path in G between a and b . We first prove that there is always some vertex $v \in A \cup R^{\leq k}$ (a set known to our algorithm) for which $p_v(A)$ is large.

▷ **Claim 4.** We have $p := \max_{v \in A \cup R^{\leq k}} p_v^G(A) \geq \frac{1}{2(\Delta^k + 1)}$.

Proof. Our original tree decomposition also restricts to a tree decomposition for $G[A]$, so Lemma 3 shows that there exists a bag B of T such that B is a $\frac{1}{2}$ -balanced separator of $G[A]$. Note that $G[A]$ is connected, so there exists some $a \in A \cap B$. As T is a witness of G being of bounded treelength, the distance between any two vertices of B is at most k . In particular, $B \subseteq N^{\leq k}[a] \subseteq A \cup R^{\leq k}$, and $|B| \leq \Delta^k + 1$ since G has maximum degree Δ . Moreover, since B is a $\frac{1}{2}$ -balanced separator of $G[A]$, for at least half of the pairs $\{u, v\} \subseteq A$, the shortest path between u and v goes through B . Using the pigeonhole principle, there exists a $v \in B$ such that $p_v^G(A) \geq \frac{1}{2(\Delta^k + 1)}$. ◁

The next three claims are building on each other to find a balanced separator S . In the first one, we argue that we can find, using few queries, a vertex with high betweenness.

▷ **Claim 5.** There is a randomised algorithm that finds $z \in N^{\leq 3k/2}[A]$ with $p_z^G(A) \geq p/2$ with probability at least $2/3$ using $O(p^{-1}(n_A + r) \log(n_A + r))$ distance queries in G .

Proof. To simplify notation, we omit G and A from $p_v^G(A)$ and only write p_v . We first sample uniformly and independently (with replacement) pairs of vertices $\{u_i, v_i\} \subseteq A$ for $i \in [C \log(n_A + r)]$ where $C \leq \frac{1}{2p} + 1$ is defined later. Then, we ask $\text{QUERY}(u_i, N^{\leq 3k/2}[A])$ and $\text{QUERY}(v_i, N^{\leq 3k/2}[A])$.

We write

$$P_i = \{x \in N^{\leq 3k/2}[A] \mid d(u_i, x) + d(x, v_i) = d(u_i, v_i)\}$$

for the set of vertices that are on a shortest path between u_i and v_i . Note that Lemma 2 implies that P_i contains all vertices of $V(G)$ on a shortest path from u_i to v_i . From the queries done above we can compute P_i for all $i \in [C \log(n_A + r)]$. For each vertex $v \in N^{\leq 3k/2}[A]$, we denote by \tilde{p}_v an estimate of p_v defined by $\tilde{p}_v = |\{i \in [C \log(n_A + r)] : v \in P_i\}| / (C \log(n_A + r))$. The algorithm outputs z such that $z = \arg \max_{v \in N^{\leq 3k/2}[A]} \tilde{p}_v$.

The query complexity of this algorithm is $2C \log(n_A + r) |N^{\leq 3k/2}[A]| = O_{k, \Delta}(n_A \log(n_A + r))$.

We now justify the correctness of this algorithm and give C . Let $y = \arg \max_{w \in N^{\leq 3k/2}[A]} p_w$. We need to show that $p_z \leq \frac{p_y}{2}$ has probability at most $\frac{1}{3}$. Let u be a vertex chosen uniformly at random among the set of vertices $w \in N^{\leq 3k/2}[A]$ with $p_w \leq p_y/2$. A simple union bound implies that it is sufficient to show that $\mathbb{P}[\tilde{p}_y \leq \tilde{p}_u] < 1/(3n_A + 3r)$. Indeed, this implies that the probability that a vertex w with $p_w \leq p_y/2$ is a better candidate for z than y , is at most $1/3$. Note that the elements of $\{\tilde{p}_w \mid w \in N^{\leq 3k/2}[A]\}$ (and thereby z) are random variables depending on the pairs of vertices sampled at the start, and that the elements of $\{p_w \mid w \in N^{\leq 3k/2}[A]\}$ are fixed.

We denote by A_i the event $\{u \in P_i\}$ and by B_i the event $\{y \in P_i\}$. The events $(A_i)_i$ are independent, since each pair $\{u_i, v_i\}$ has been sampled uniformly at random and independently. By definition, $\mathbb{P}[A_i] = p_u \leq p_y/2$ and $\mathbb{P}[B_i] = p_y$. Thus, the random variable X_i defined by $X_i = \mathbb{1}_{A_i} - \mathbb{1}_{B_i}$ has expectation $\mathbb{E}[X_i] \leq -p_y/2$. Therefore, applying Hoeffding's inequality [8], we obtain

$$\mathbb{P}\left[\sum_{i=1}^{C \log(n_A + r)} X_i \geq 0\right] \leq 2 \exp\left(-\frac{2(C \log(n_A + r)p_y/2)^2}{4 \log(n_A + r)}\right).$$

By choosing $\frac{1}{2p} + 1 \geq C \geq \frac{1}{2p_y} = \frac{1}{2p}$ such that $C \log(n_A + r)$ is an integer, we conclude that

$$\mathbb{P}[\tilde{p}_y \leq \tilde{p}_u] = \mathbb{P}\left[\sum_{i=1}^{C \log(n_A + r)} X_i \geq 0\right] \leq 2 \exp(-2 \log(n_A + r)) \leq 1/(3n_A + 3r)$$

for $n_A \geq 6$. This completes the proof. \triangleleft

Let z be a vertex with high betweenness as in the claim above. We now argue that $N^{3k/2}[z]$ is an α -balanced separator for some constant α depending only on Δ and k .

\triangleright **Claim 6.** Let $\alpha = \sqrt{1 - \frac{1}{4(\Delta^k + 1)}}$. If $z \in N^{\leq 3k/2}[A]$ satisfies $p_z^G(A) \geq p/2$, then $S := N^{\leq 3k/2}[z]$ is an α -balanced separator for A .

Proof. Suppose towards contradiction that S is not an α -balanced separator. Thus there is a connected component C of $G[V(G) \setminus S]$ with $|C \cap A| > \alpha n_A$. By definition of S , $d(z, C) > 3k/2$ which implies by Lemma 2 that for any pair of vertices in C , no shortest path between these two vertices goes through z . In particular, this holds for pairs of vertices in $C \cap A$. Therefore,

$$p_z^G(A) \leq \frac{(n_A^2 - |C \cap A|^2)}{n_A^2} < 1 - \alpha^2 = 1 - \left(1 - \frac{1}{4(\Delta^k + 1)}\right) = \frac{1}{4(\Delta^k + 1)} \leq p/2$$

using Claim 4 for the last step, contradicting our assumption that $p_z^G(A) \geq p/2$. \triangleleft

We apply Claim 5 to find $z \in N^{\leq 3k/2}$, where $p_z^G(A) \geq p/2$ with probability at least $2/3$ (using also Claim 4). We compute $S = N^{\leq 3k/2}[z]$ using $O_{k,\Delta}(n_A + r)$ distance queries; this can be done since $S \subseteq A \cup R^{\leq 3k}$ so the algorithm only needs to consider $n_A + r$ vertices when searching for neighbours.

The set S is an α -balanced separator with probability at least $2/3$ by Claim 6. In particular, the algorithm does not know yet at this point if it is indeed a good separator or not. It will be able to determine this after computing the partition of $G[A \setminus S]$ into connected components.

The following claim uses mostly the same algorithm as [9, Alg. 6], and the proof is analogous. As we are using this algorithm in a slightly different setting, we still give a complete proof of the lemma.

▷ **Claim 7.** There is a deterministic algorithm that given a set $S \subseteq A$, computes the partition of $A \setminus S$ into connected components of $G[A \setminus S]$ using at most $n_A \cdot \Delta(r + |S|)$ distance queries.

Proof. By assumption, R^1 is the set of vertices at distance exactly 1 from A in G . Since A is connected, it is a connected component of $G[V(G) \setminus R^1]$. Therefore, the connected components of $G[A \setminus S]$ are exactly the connected components of $G[V(G) \setminus (R^1 \cup S)]$ containing an element of A . We denote by B the open neighbourhood of $S \cup R^1$ in A , that is, $B = (N[S \cup R^1] \cap A) \setminus (S \cup R^1)$. We use the following algorithm.

- We ask $\text{QUERY}(A, S \cup R^1)$ in order to deduce $N[S \cup R^1] \cap A$, and then we ask $\text{QUERY}(A, N[S \cup R^1] \cap A)$.
- We compute $D_b = \{v \in A \cap S \mid d(v, b) \leq d(v, S \cup R^1)\}$ for $b \in B$, the set of vertices in $A \cap S$ which have a shortest path to b that does not visit a vertex of $S \cup R^1$.
- Let $\mathcal{D} = \{D_s \mid s \in B\}$. While there are two distinct elements $D_1, D_2 \in \mathcal{D}$ such that $D_1 \cap D_2 \neq \emptyset$, merge them in \mathcal{D} , that is, update $\mathcal{D} \leftarrow (\mathcal{D} \setminus \{D_1, D_2\}) \cup \{D_1 \cup D_2\}$. We output \mathcal{D} .

Note that any vertex $a \in A \setminus S$, is not in $S \cup R^1$, so will be in D_s for at least one $s \in B$ (possibly $s = a$) before we do the last step of the algorithm. The last step ensures that the output is indeed a partition of A .

We first argue that \mathcal{D} , as outputted by the algorithm above, is an over-approximation of the connected component partition of $G[A \setminus S]$ (that is, for any connected component C of $G[A \setminus S]$, there exists $D \in \mathcal{D}$ such that $C \subseteq D$). It suffices to prove that for any edge $ab \in E(G[A \setminus S])$ there exists $D \in \mathcal{D}$ such that $\{a, b\} \subseteq D$. Suppose without loss of generality that $d(a, S \cup R^1) \leq d(b, S \cup R^1)$. Moreover let $s \in B$ such that $d(a, s) = d(a, S \cup R^1) - 1$ and thus $a \in D_s$. Now $d(b, s) \leq d(a, s) + 1 \leq d(b, S \cup R^1)$ thus $b \in D_s$.

We now argue that \mathcal{D} is an under-approximation too, by showing that $G[D \setminus S]$ is connected for all $D \in \mathcal{D}$. We first show this for the initial sets D_s with $s \in N[S \cup R^1] \cap A$. Let $s \in B$. For any $v \in D_s$, by definition, $d(v, s) \leq d(v, S \cup R^1)$, thus there is a shortest path P between v and s not using vertices of $S \cup R^1$. Moreover $s \in A$ and A is separated from $V(G) \setminus A$ by R^1 , therefore P is contained in $A \setminus S$. This shows that v is in the same connected component of $G[A \setminus S]$ as s . To see that $G[D]$ remains connected for all $D \in \mathcal{D}$ throughout the algorithm, note that when the algorithm merges two sets $D_1, D_2 \in \mathcal{D}$, they need to share a vertices, thus if both $G[D_1]$ and $G[D_2]$ are connected then $G[D_1 \cup D_2]$ is also connected.

Remember that $|S| \leq \Delta^{3k/2} + 1 = O_{k,\Delta}(1)$ and that the bounded degree condition implies $|N[S \cup R^1]| \leq \Delta \cdot |S \cup R^1|$. This allow us to conclude that the query complexity is bounded by

$$|A| \cdot |N[S \cup R^1]| \leq n_A \cdot \Delta |S \cup R^1| \leq n_A \cdot \Delta(r + |S|). \quad \blacktriangleleft$$

We apply the algorithm from Claim 7 with the separator S computed by Claim 6. Knowing the partition, the algorithm can check if S is indeed α -balanced. If not, the algorithm repeats Claim 7 and computes a new potential separator. An single iteration succeeds with probability at least $2/3$ and each iteration is independent from the others, so the expected number of repetitions is $3/2$.

We ask $\text{QUERY}(S \cup R^{\leq 3k}, A)$. For each connected component \tilde{A} of $G[A \setminus S]$, we will reconstruct $G[\tilde{A}]$ and then we will describe how to reconstruct $G[A]$. If $|\tilde{A}| \leq \log(n)$, then we ask $\text{QUERY}(\tilde{A}, \tilde{A})$ to reconstruct $G[\tilde{A}]$. Otherwise, we will place a recursive call on \tilde{A} , after guaranteeing that our desired properties mentioned at the start are again satisfied. By definition, $G[\tilde{A}]$ is connected. So we know property 1 holds when A is replaced by \tilde{A} .

To ensure properties 2 and 3 are also satisfied for the recursive call, we reconstruct \tilde{R}^i , the set of vertices at distance exactly i from \tilde{A} . As $S \cup R^1$ separates \tilde{A} from other component of $G[A \setminus S]$, for any other connected component D of $G[A \setminus S]$ and for any $v \in D$, we have:

$$d(\tilde{A}, v) = \min_{s \in S \cup R^1} d(\tilde{A}, s) + d(s, v).$$

Therefore we can compute $d(\tilde{A}, v)$ from the query results of $\text{QUERY}(S \cup R^{\leq 3k}, A)$ for all $v \in A \cup R^{\leq 3k}$. This is enough to deduce \tilde{R}^i for any $i \leq 3k$ because $\tilde{A} \subseteq A$ and thus $\tilde{R}^i \subseteq A \cup R^i$.

After we have (recursively) reconstructed $G[\tilde{A}]$ for each connected component \tilde{A} of $G[A \setminus S]$, we reconstruct $G[A]$ by using that we already know all the distance between all pairs (a, s) with $a \in A$ and $s \in S$. In particular, as we already asked $\text{QUERY}(S \cup R^{\leq 3k}, A)$ earlier in the algorithm, we know $G[S \cap A]$ and also how to “glue” the components to this (namely, by adding edges between vertices at distance 1).

By Claim 6, each recursive call reduces the size of the set A under consideration by a multiplicative factor of α . Therefore, the recursion depth is bounded by $O_{\Delta, k}(\log n)$ and the algorithm will terminate.

We argued above that the algorithm correctly reconstructs the graph. It remains to analyse the query complexity.

We analyse the query complexity via the recursion tree, where we generate a child for a vertex when it places a recursive call. We can associate to each vertex v of the recursion tree T_R , a subset $A_v \subseteq V(G)$ for which the algorithm is trying to reconstruct $G[A_v]$. The subsets associated to the children of a node v are disjoint, since each corresponds to a connected component of $A_v \setminus S_v$ for some subset $S_v \subseteq V(G)$ that is an α -balanced separator. In particular, the subsets associated to the leafs are disjoint.

In a leaf node v , the algorithm performs $|A_v|^2$ queries to reconstruct $G[A_v]$, where $|A_v| \leq \log(n)$. If we enumerate the sizes A_v for the leafs v of the recursion tree as a_1, \dots, a_ℓ , then $\sum_{i=1}^{\ell} a_i^2 \leq \ell \log(n)^2 \leq n \log(n)^2$, where we use that we have at most n leafs since the corresponding subsets are disjoint.

Since there are at most n leafs, and the recursion depth is $O_{k, \Delta}(\log n)$, there are $O_{k, \Delta}(n \log n)$ internal nodes. Let v be an internal node and let n_A and r denote the sizes of the corresponding subsets $A = A_v$ and $R^{\leq 3k}$. The algorithm makes the following queries:

- Finding z takes $O_{k, \Delta}(n_A \log(n_A + r))$ queries in Claim 5.
- $O_{k, \Delta}(n_A r)$ queries to compute S from z and to find the connected components of $A \setminus S$ in Claim 7. This step and the previous step are repeated a constant number of times (in expectation).
- $O_{k, \Delta}(n_A r)$ queries to set up the recursive calls to the children of v .

Since each recursive call increases the size of $R^{\leq 3k}$ by at most an additive constant smaller than $(\Delta + 1)^{9k/2}$ (recall that $\tilde{R}^{\leq 3k} \subseteq R^{\leq 3k} \cup N^{\leq 9k/2}[z]$), and the recursion depth is $O_{k, \Delta}(\log n)$, it follows from an inductive argument that $r = O_{k, \Delta}(\log n)$. So the number of queries listed above is $O_{k, \Delta}(n_A \log n)$.

To compute the total query complexity of internal nodes, we use the fact that for two nodes v and v' at the same recursion depth we have that $A_v \cap A_{v'} = \emptyset$. Therefore, by adding contribution layer by layer in the recursion tree we get a query complexity of $O_{k,\Delta}(n \log n)$ for any fixed layer, and the total number of queries performed sum up to:

$$n \log^2 n + O_{k,\Delta}(\log n)O_{k,\Delta}(n \log n) = O_{k,\Delta}(n \log^2 n). \quad \blacktriangleleft$$

We did not try to optimise the dependence in k and Δ hidden in the $O_{k,\Delta}$ notation throughout the proof of Theorem 1. Expanding all $O_{k,\Delta}$ notations in the proof implies that our algorithm uses $\Delta^{O(k)} n \log^2 n$ queries. It would be interesting to reduce this dependence to a polynomial in Δ and k .

4 Conclusion

In this paper, we shed further light on what graph structures allow efficient distance query reconstruction. We expect that the true deterministic and randomised query complexity of graphs of bounded treelength and bounded maximum degree is $\Theta(n \log n)$, matching the lower bound which already holds for trees from [1].

It seems natural that having small balanced separators helps with obtaining a quasi-linear query complexity. We show this is indeed the case when some additional structure on the separator is given (namely, vertices being “close”). A possible next step would be to see if this additional structure can be removed.

► **Problem 1.** *Does there exist a randomised algorithm that reconstructs an n -vertex graph of maximum degree Δ and treewidth k using $\tilde{O}_{\Delta,k}(n)$ queries in expectation?*

Some parts of the algorithm still work, such as checking whether a given set S is a balanced separator (via Claim 7). When trying to recursively reconstruct one of the components, it is important to “keep enough information about the distances”. In our algorithm, we can include the shortest paths between the vertices in the separator; this is the main purpose of the “boundary sets” R^i and why we carefully chose the domain for z in Claim 5. The possibility to do this is almost the definition of bounded treelength. Therefore, we believe that a new approach would be needed to produce a good candidate for a balanced separator in the general case.

Finally, we remark that it may very well be that techniques building on separators are needed as part of a potential quasi-linear algorithm for reconstructing graph classes that do not directly guarantee the existence of such separators. Indeed, there are approaches that actually do not work well even on trees, yet are good at handling certain graphs without small balanced separators, and perhaps a combination of both types of methods will be needed to handle the class of all bounded degree graphs. For example, the approach taken by [12] is to ask all queries to a randomly selected set of vertices. On some graph classes (such as random regular graphs, which do not have small balanced separators), this already forces most of the non-edges with high probability and so the remaining pairs can be queried directly. But in order to beat the best-known upper bound for general graphs of bounded degree (of $\tilde{O}_{\Delta}(n^{3/2})$ from [11]), such an approach cannot be applied directly, even for trees. Indeed, for a complete binary tree on n vertices, the distances to any set S with at most $\frac{1}{100}\sqrt{n}$ vertices, no matter how cleverly chosen, leave many pairs of distances undetermined. In fact, there are approximately \sqrt{n} vertices at height $\lfloor \frac{1}{2} \log n \rfloor$ in this tree, and S will miss the “trees below” most of those \sqrt{n} vertices entirely. This means that there are still $\Omega(n^{3/2})$ pairs u, v that form a non-edge, yet have the same distance to all vertices in S . This means that even for the class of all bounded degree graphs, there may need to be a part of the algorithm which exploits the structure of “nice” separators, when they exist.

References

- 1 Paul Bastide and Carla Groenland. Optimal distance query reconstruction for graphs without long induced cycles. *arXiv preprint*, 2023. doi:10.48550/arXiv.2306.05979.
- 2 Zuzana Beerliova, Felix Eberhard, Thomas Erlebach, Alexander Hall, Michael Hoffmann, Mat Mihal'ak, and L Shankar Ram. Network discovery and verification. *IEEE Journal on selected areas in communications*, 24(12):2168–2181, 2006. doi:10.1109/JSAC.2006.884015.
- 3 Rémy Belmonte, Fedor V Fomin, Petr A Golovach, and MS Ramanujan. Metric dimension of bounded tree-length graphs. *SIAM Journal on Discrete Mathematics*, 31(2):1217–1243, 2017. doi:10.1137/16M1057383.
- 4 Eli Berger and Paul Seymour. Bounded-diameter tree-decompositions. *arXiv:2306.13282*, 2023.
- 5 Reinhard Diestel and Malte Müller. Connected tree-width. *Combinatorica*, 38(2):381–398, 2018. doi:10.1007/S00493-016-3516-5.
- 6 Thomas Dissaux, Guillaume Ducoffe, Nicolas Nisse, and Simon Nivellet. Treelength of series-parallel graphs. *Procedia Computer Science*, 195:30–38, 2021. doi:10.1016/J.PROCS.2021.11.008.
- 7 Yon Dourisboure and Cyril Gavaille. Tree-decompositions with bags of small diameter. *Discrete Mathematics*, 307(16):2008–2029, 2007. doi:10.1016/J.DISC.2005.12.060.
- 8 Wassily Hoeffding. Probability inequalities for sums of bounded random variables. In *The collected works of Wassily Hoeffding*, pages 409–426. Springer, 1994.
- 9 Sampath Kannan, Claire Mathieu, and Hang Zhou. Near-linear query complexity for graph inference. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 773–784, 2015. doi:10.1007/978-3-662-47672-7_63.
- 10 Adrian Kosowski, Bi Li, Nicolas Nisse, and Karol Suchan. k -chordal graphs: From cops and robber to compact routing via treewidth. *Algorithmica*, 72(3):758–777, 2015. doi:10.1007/S00453-014-9871-Y.
- 11 Claire Mathieu and Hang Zhou. Graph reconstruction via distance oracles. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 733–744, 2013. doi:10.1007/978-3-642-39206-1_62.
- 12 Claire Mathieu and Hang Zhou. A simple algorithm for graph reconstruction. *Random Structures & Algorithms*, pages 1–21, 2023. doi:10.1002/rsa.21143.
- 13 Lev Reyzin and Nikhil Srivastava. Learning and verifying graphs using queries with a focus on edge counting. In *Algorithmic Learning Theory: 18th International Conference, ALT 2007, Sendai, Japan, October 1-4, 2007. Proceedings 18*, pages 285–297. Springer, 2007. doi:10.1007/978-3-540-75225-7_24.
- 14 Neil Robertson and Paul Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of algorithms*, 7(3):309–322, 1986. doi:10.1016/0196-6774(86)90023-4.
- 15 Guozhen Rong, Wenjun Li, Yongjie Yang, and Jianxin Wang. Reconstruction and verification of chordal graphs with a distance oracle. *Theoretical Computer Science*, 859:48–56, 2021. doi:10.1016/J.TCS.2021.01.006.

Component Order Connectivity Admits No Polynomial Kernel Parameterized by the Distance to Subdivided Comb Graphs

Jakob Greilhuber ✉ 

TU Wien, Austria

CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
Saarbrücken Graduate School of Computer Science, Germany

Roohani Sharma ✉ 

Department of Informatics, University of Bergen, Norway

Abstract

In this paper we show that the d -COMPONENT ORDER CONNECTIVITY (d -COC) problem parameterized by the distance to subdivided comb graphs (**dist-to-subdivided-combs**) does not admit a polynomial kernel, unless $\text{NP} \subseteq \text{coNP/poly}$.

The d -COC problem is a generalization of the classical VERTEX COVER problem. An instance of the d -COC problem consists of an undirected graph G and a positive integer k , and the question is whether there exists a set $S \subseteq V(G)$ of size at most k , such that each connected component of $G - S$ contains at most d vertices. When $d = 1$, d -COC is the VERTEX COVER problem.

VERTEX COVER is a ubiquitous problem in parameterized complexity, and it admits a kernel with $O(k^2)$ edges and $O(k)$ vertices, which is tight [Dell & van Melkebeek, JACM 2014]. Our result is inspired by the work of Jansen & Bodlaender [TOCS 2013], who gave the first polynomial kernel for VERTEX COVER where the parameter is provably smaller or equal to the standard parameter, solution size k . They used **fvs**, the feedback vertex set number, as the parameter. In this work, we show that unlike most other existing results or techniques for kernelization of VERTEX COVER that generalize to d -COC, this is not the case when **dist-to-subdivided-combs**, which is at least as large as **fvs**, is the parameter.

Our lower bound is achieved in two stages. In the first stage we extend the result of Hols, Kratsch & Pieterse [SIDMA 2022] where they show that if a graph family \mathcal{C} , which is closed under taking disjoint unions, has unbounded “blocking set” size, then VERTEX COVER does not admit a polynomial kernel parameterized by the size of a vertex modulator to \mathcal{C} , unless $\text{NP} \subseteq \text{coNP/poly}$. We show that a similar sufficient condition for proving the non-existence of polynomial kernels also holds for d -COC. In the second stage, we show that when \mathcal{C} is the family of subdivided comb graphs, contrary to VERTEX COVER, where the size of minimal blocking sets of graphs in \mathcal{C} is at most two [Jansen & Bodlaender, STACS 2011], the size of minimal blocking sets of graphs in \mathcal{C} for the d -COC problem can be arbitrarily large. This yields the desired lower bound. In addition to this we also show that when \mathcal{C} is a class of paths, then it still has blocking sets of size at most two for d -COC, indicating that polynomial kernels might be achievable when the parameter is the size of a vertex modulator to the class of disjoint unions of paths (linear forests).

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Component Order Connectivity, Kernelization, Structural Parameterizations, Feedback Vertex Set, Vertex Cover, Blocking Sets, Subdivided Comb Graphs

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.21

Acknowledgements The majority of the work was done at the Max Planck Institute of Informatics during a summer internship of Jakob Greilhuber. We thank the anonymous reviewers that reviewed a previous version of this paper. The presentation of this work is based on their comments. Jakob Greilhuber thanks the organizers of ALGO 2024 for their financial support.



© Jakob Greilhuber and Roohani Sharma;

licensed under Creative Commons License CC-BY 4.0

19th International Symposium on Parameterized and Exact Computation (IPEC 2024).

Editors: Édouard Bonnet and Paweł Rzażewski; Article No. 21; pp. 21:1–21:17

Leibniz International Proceedings in Informatics



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

1 Introduction

In this work we explore kernelization with respect to structural parameterizations for a well-studied generalization of the ubiquitous VERTEX COVER problem. In the VERTEX COVER problem the input is an undirected graph G and the goal is to find a minimum set of vertices of G , say S , such that $G - S$ has no edges, or equivalently, every connected component of G contains only a single vertex. VERTEX COVER has garnered a lot of attention since the advent of parameterized complexity [17, 36, 6, 10, 19, 7, 13, 39, 33, 35, 28, 1, 2, 9, 16, 37, 38, 40, 8, 24, 20, 5, 26, 4] and has resulted in the inception of new tools and ideas that have proved useful for several other problems as well [6, 10, 19, 7, 31, 32].

Component Order Connectivity. One of the problems that has been at the receiving end of applications and generalizations of the tools developed for VERTEX COVER (especially in the context of parameterized algorithms and kernelization), is the d -COMPONENT ORDER CONNECTIVITY (d -COC) problem. In this problem, d is a fixed integer, the input is an undirected graph and the goal is to find a minimum set of vertices whose deletion results in a graph in which every connected component has at most d vertices. Note that 1-COC is precisely the VERTEX COVER problem.¹ VERTEX COVER parameterized by the solution size k admits a kernel with $O(k^2)$ edges and $O(k)$ vertices, which is tight [14]. Starting from the Buss kernelization [6], all the way up to crown decomposition based rules [10, 19] or the (Weighted) Expansion Lemma [32], and the LP-based reduction rules using the Nemhauser-Trotter theorem [7, 31, 32]; all these techniques for getting a polynomial kernel for VERTEX COVER parameterized by the solution size extend beautifully to the d -COC problem parameterized by the solution size [18, 32, 41, 21].

Amidst the kernelization results for VERTEX COVER parameterized by the solution size, Jansen and Bodlaender [28] initiated the study of kernelization for VERTEX COVER with a refined parameter, that is at most as large as the solution size. In particular, they studied the size of a minimum feedback vertex set of the graph, denoted by fvs , as the parameter, and proved that the problem admits a polynomial kernel. Observe that fvs could potentially be much smaller than the size of a minimum vertex cover. The work by Jansen and Bodlaender [28] led to several new and interesting insights about the problem. A next natural question is: can the result of Jansen and Bodlaender [28] be extended to the d -COC problem? In work that is independent of ours, this relevant question was negatively answered by Donkers and Jansen [15], unless $\text{NP} \subseteq \text{coNP/poly}$, no polynomial kernel exists for d -COC parameterized by fvs , for all $d \geq 2$. We study the problem with a parameter that is at least as large as fvs , the *distance to subdivided comb graphs* ($\text{dist-to-subdivided-combs}$). We prove that for all integers $d \geq 2$, even when parameterizing by the distance to subdivided comb graphs, d -COC does not admit a polynomial kernel, unless $\text{NP} \subseteq \text{coNP/poly}$. Note that, while our result is not explicitly stated as a result in [15], it may be possible to also obtain it from their proofs with some careful considerations.

For completeness, we also remark that Courcelle's theorem (see for example [12, Section 7.4] or [11]) can be used to show that the problem is FPT, even when parameterizing by fvs . Besides our work and the work by Donkers and Jansen [15] there are some other publications that study d -COC kernelization with structural parameters. For instance, Bhyravarapu et al. [3] parameterize the problem by the size of a set that is a solution to the c -COC problem, where $c \geq d$, and Jansen and Pieterse [29] study a generalization of the problem parameterized by the size of a treedepth- η modulator.

¹ Note that other than being a generalization of VERTEX COVER, d -COC is an interesting problem in its own right. See [23].

1.1 Our results and methods

As mentioned above, we show that d -COC does not admit a polynomial kernel parameterized by the *distance to subdivided comb graphs*. A *comb graph* is a graph that consists of a central path and for each vertex of this central path, there is a unique pendant vertex adjacent to it. A *subdivided comb graph* is obtained from a comb graph by repeatedly subdividing edges. The *distance to subdivided comb graphs* is the smallest integer k such that there is a vertex set of size k whose deletion results in a disjoint union of subdivided comb graphs. Therefore, distance to subdivided comb graphs is at least as large as **fvs**. The *hairlength* of a subdivided comb graph is the maximum number of vertices of any path attached to the central path minimized over all possible choices for the central path.

► **Theorem 1.** *For any integer $d \geq 2$, d -COMPONENT ORDER CONNECTIVITY does not admit a polynomial compression when the parameter is distance to subdivided comb graphs, unless $\text{NP} \subseteq \text{coNP/poly}$.*

Note that showing the non-existence of a polynomial compression is stronger than showing the non-existence of a polynomial kernel (see Section 2).

We prove Theorem 1 by generalizing the notion of *blocking sets*, which was formally introduced by Hols, Kratsch & Pieterse [26] for the VERTEX COVER problem,² to d -COC. A blocking set of a graph G , for VERTEX COVER, is a subset Y of the vertices of G , such that no minimum vertex cover contains Y . Hence, a blocking set “blocks” itself from being extended into an optimal solution. Of special relevance are *minimal* blocking sets, which are blocking sets such that no strict subset of them is a blocking set. This concept of (minimal) blocking sets can be generalized to d -COC for any integer $d \geq 2$ in the natural way. Concretely, define a *d -blocking set* of G as a subset Y of the vertices of G such that there is no minimum solution of d -COC that contains Y .

One of the key results by Hols et al. [26] gives a sufficient condition for proving kernelization lower bounds for structural parameterizations for VERTEX COVER. They show that if a graph class \mathcal{C} , which is closed under taking the disjoint union, contains a graph with a minimal blocking set of size α , then no kernel of size $\mathcal{O}(|X|^{\alpha-\varepsilon})$ can exist for all $\varepsilon > 0$, when parameterizing by the size of a smallest vertex modulator X to \mathcal{C} , unless $\text{NP} \subseteq \text{coNP/poly}$. Hence, when the size of minimal blocking sets of graphs in \mathcal{C} is unbounded, then one can rule out a polynomial kernel altogether parameterized by $|X|$.

We extend this result by Hols et al. [26] to the d -COC problem, showing that a similar property, with minimal d -blocking sets, holds for d -COC, for all integers $d \geq 1$ (Theorem 4). We later show that even though the same condition implies kernelization lower bounds for both VERTEX COVER and d -COC, the family of forests behaves differently for VERTEX COVER and d -COC: no forest has a minimal 1-blocking set of size more than two (implicitly proved in [27], the conference version of [28]), whereas there are forests (in fact subdivided comb graphs) with arbitrarily large minimal d -blocking sets for every $d \geq 2$ (Lemma 5).

For the first part (as stated above), we provide a linear parameter transformation from the α -HITTING SET problem, which is the classical HITTING SET problem in which every set that needs to be hit has cardinality exactly α , and the size of the universe \mathcal{U} is the parameter. For this problem, it is known that no polynomial compression with size $\mathcal{O}(|\mathcal{U}|^{\alpha-\varepsilon})$ exists for all $\varepsilon > 0$ and all integers $\alpha \geq 2$, unless $\text{NP} \subseteq \text{coNP/poly}$ [14, Theorem 2].

² The notion of blocking sets had (implicitly) appeared in a lot of other kernelization results for VERTEX COVER like [28, 30, 22, 34, 5, 3].

In the parameter transformation, one creates a “universe vertex” for each element of the universe of the input α -HITTING SET instance. Then, for each set F that needs to be hit in the input instance, one creates multiple copies of a graph H of \mathcal{C} that has a minimal d -blocking set of size α . Each such copy is associated with the set F , and, each vertex of the minimal d -blocking set is associated with a different element of F . Then, each vertex of the minimal d -blocking set is connected to the universe vertex corresponding to the associated element. This way, it is ensured that the set of all universe vertices is a modulator to \mathcal{C} , and it has the same size as the universe of the input α -HITTING SET instance. Moreover, if the bound on the size of the solution is carefully chosen, one can ensure that the d -COC instance is a yes-instance if and only if the input instances is a yes-instance. The intuition behind this is that a solution to the input instance can be easily mapped to a small d -COC solution by choosing the universe vertices of the hitting set, and by extending this selection to minimum solutions within the copies of H . On the other hand, if there is a small d -COC solution, it must be the case that the selection within the universe vertices corresponds to a small hitting set of the input instance. Otherwise, one would necessarily have to put all vertices of the minimal d -blocking set into the solution for many copies of H , which would make the solution too large.

To complete the proof of Theorem 1, for all integers $N \geq 1, d \geq 2$, we then define a blocking set graph $T_{N,d}$, which is a subdivided comb graph with hairlength at most $d + 1$, such that this graph has a minimal d -blocking set of size N . Note that providing graphs with large d -blocking sets is easy, the difficulty of the construction stems from the fact that the sets must also be minimal. Combined with the previous result, this proves that when \mathcal{C} is a graph class closed under taking the disjoint union that contains the blocking set graph $T_{N,d}$ for arbitrarily large N , then d -COC parameterized by the size of a smallest modulator to \mathcal{C} does not have a polynomial compression, unless $\text{NP} \subseteq \text{coNP/poly}$.

Contrasting the fact that the class of subdivided comb graphs has minimal d -blocking sets of unbounded size when $d \geq 2$, we show that this is not the case for path graphs. We show that for paths, any minimal blocking set has size at most two. Hence, there is still hope for obtaining a polynomial kernel when \mathcal{C} only contains disjoint unions of paths.

2 Preliminaries

For $n, m \in \mathbb{Z}$, by $[n, m]$ we denote the set $\{x \in \mathbb{Z} \mid n \leq x \leq m\}$. For $n, m \in \mathbb{Z}, k \in \mathbb{Z}^+$, we write $\{n, n + k, \dots, m\}$ for the set $\{x \in \mathbb{Z} \mid n \leq x \leq m, x \equiv n \pmod{k}\}$. In particular, $\{0, 2, \dots, 0\} = \{0\}$.

Graphs. A graph G is a pair (V, E) , where V is a set and $E \subseteq \binom{V}{2}$. For a graph G , we also refer to the vertex set of G as $V(G)$ and the edge set of G as $E(G)$. In this work all considered graphs are finite, undirected, and contain no multi-edges or self-loops. For a graph G and $v \in V(G)$, the (open) neighbourhood of v is defined as $N_G(v) := \{u \mid vu \in E(G)\}$. We may omit the subscript G if the used graph is clear from the context. Graph H is a subgraph of graph G if and only if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. For a graph G and vertex set $X \subseteq V(G)$, $G[X]$ denotes the graph induced by X , that is, $G[X] = H$, where H is the subgraph of G with $V(H) = X$ and $E(H) = \{uv \mid uv \in E(G), u \in X, v \in X\}$. For graph G and $X \subseteq V(G)$, we write $G - X$ for the graph $G[V(G) \setminus X]$. Given a graph G and edge $uv \in E(G)$, the operation of subdividing edge uv results in the graph G' with $V(G') = V(G) \cup \{w\}$ and $E(G') = (E(G) \setminus \{uv\}) \cup \{uw, vw\}$, where w is a new vertex not in $V(G)$.

Kernelization and compression. A parameterized problem Q is a subset of $\Sigma^* \times \mathbb{N}$ for a finite alphabet Σ . An instance $(I, k) \in \Sigma^* \times \mathbb{N}$ of problem Q is a yes-instance if $(I, k) \in Q$, and a no-instance otherwise. The value k of instance (I, k) is the parameter of the instance. A *kernelization algorithm* takes an instance (I, k) of a parameterized problem Q as input, and outputs an equivalent instance (I', k') of Q such that $|I'| + k' \leq f(k)$ for some computable function f . The runtime of the algorithm must be polynomial in $|I| + k$. Such an algorithm is often simply called a *kernel*, and a *polynomial kernel* if f is a polynomial function. A *polynomial compression* is defined similarly to kernelization, except that the output instance I' may be of a different unparameterized problem P . Note that if a problem has a polynomial kernel, it also has a polynomial compression, and hence, ruling out polynomial compressions also rules out polynomial kernels. We refer to the textbooks [12, 21] for more details on these concepts.

Kernelization lower bounds. In this paper, we use a well-established technique for obtaining kernelization lower bounds. It resembles classical reductions, and is fittingly called *linear parameter transformation* (see [25]).

► **Definition 2** (linear parameter transformation). *Let P and Q be parameterized problems. A linear parameter transformation is an algorithm that transforms an instance (I, k) of P into an equivalent instance (I', k') of Q in time polynomial in $|I| + k$, such that $k' \in O(k)$.*

When Q has a polynomial compression of size $O(p^\alpha)$ for parameter p , and there is linear parameter transformation from P with parameter k to Q with parameter p , then by using the transformation, one obtains a polynomial compression for P of size $O(k^\alpha)$. Hence, if we know that P does not have a polynomial compression of size $O(k^\alpha)$, then neither can Q of size $O(p^\alpha)$.

3 Large minimal blocking sets as bottlenecks for polynomial kernels for d -COC

In this section, we extend the sufficient condition for proving kernelization lower bounds for VERTEX COVER given by Hols et al. [26], to the d -COC problem. Concretely, we will define the notion of minimal blocking sets for d -COC, and we will show that no graph class, that is closed under taking disjoint unions and has blocking sets of unbounded size, admits a polynomial compression, unless $\text{NP} \subseteq \text{coNP/poly}$. We begin by stating the definition of blocking sets [26, Definition 2.3] for d -COC. For a graph G , we call $S \subseteq V(G)$ a *d -coc set* of G if every connected component of $G - S$ has at most d vertices.

► **Definition 3** (Blocking sets). *Let G be a graph and d be a positive integer. A set $Y \subseteq V(G)$ is a d -blocking set if Y is not a subset of any minimum d -coc set of G . A d -blocking set Y is a minimal d -blocking set if no set $Y' \subsetneq Y$ is a d -blocking set. That is, for every $y \in Y$, there exists a minimum d -coc set of G that contains $Y \setminus \{y\}$.*

Given a graph class \mathcal{C} and a graph G , a (vertex) modulator to \mathcal{C} is a set $X \subseteq V(G)$ such that $G - X$ is in \mathcal{C} . Hols et al. [26, Theorem 1.1] show that if \mathcal{C} is a graph class that is closed under taking disjoint unions, and \mathcal{C} contains a graph with a minimal 1-blocking set of size α , then, VERTEX COVER parameterized by the size of a vertex modulator X to \mathcal{C} does not have a kernel of size $\mathcal{O}(|X|^{\alpha-\varepsilon})$ for all $\varepsilon > 0$, unless $\text{NP} \subseteq \text{coNP/poly}$. This powerful theorem allows one to derive kernelization lower bounds for various structural parameterizations in a convenient way.

We now extend the above result to the d -COC problem. The proof we present is inspired by that of Hols et al. [26, Theorem 1.1]. We mention the key difference between our reduction and the reduction by Hols et al. after presenting our reduction.

The proof is done via a linear parameter transformation from α -HITTING SET parameterized by the universe size, which is the problem defined below, to the d -COC problem parameterized by the size of a vertex modulator to a graph class \mathcal{C} that is closed under taking the disjoint union and that contains a graph with a minimal d -blocking set of size $\alpha \geq 2$. The α -HITTING SET problem is the same as the classical HITTING SET problem, except that every set that needs to be hit has cardinality exactly α , and that the parameter is the size of the universe.

α -HITTING SET

Input: Universe \mathcal{U} , family of subsets $\mathcal{F} \subseteq 2^{\mathcal{U}}$ such that $|F| = \alpha$ for all $F \in \mathcal{F}$, integer $k \geq 0$

Parameter: $|\mathcal{U}|$

Question: Is there a set $H \subseteq \mathcal{U}$ such that, for all $F \in \mathcal{F}$, $H \cap F \neq \emptyset$ and $|H| \leq k$?

A lower bound for α -HITTING SET given by Dell and van Melkebeek [14, Theorem 2] shows that there is no polynomial compression of size $\mathcal{O}(|\mathcal{U}|^{\alpha-\varepsilon})$ for α -HITTING SET when $\alpha \geq 2$, for any $\varepsilon > 0$, unless $\text{NP} \subseteq \text{coNP/poly}$. We denote the problem d -COC where the parameter is the size of a modulator to \mathcal{C} , for some graph class \mathcal{C} , as d -COC/dist-to- \mathcal{C} . Note that we assume that a modulator to \mathcal{C} is given together with the input.

► **Theorem 4.** *Let \mathcal{C} be a class of graphs that is closed under disjoint union, and d be a positive integer. If there is a graph in \mathcal{C} with a minimal d -blocking set of cardinality $\alpha \geq 2$, then there exists no polynomial compression for d -COC/dist-to- \mathcal{C} of size $\mathcal{O}(|X|^{\alpha-\varepsilon})$, where X is the modulator to \mathcal{C} provided with the input, for all $\varepsilon > 0$, unless $\text{NP} \subseteq \text{coNP/poly}$.*

Proof. Assume that there exists a graph H in \mathcal{C} such that H has a minimal d -blocking set of size $\alpha \geq 2$. Moreover, let OPT denote the size of a minimum d -coc set of H . We prove the polynomial compression lower bound by a linear parameter transformation from α -HITTING SET. Let $(\mathcal{U}, \mathcal{F}, k)$ be the input instance of α -HITTING SET. If $k \geq |\mathcal{U}|$, we have a trivial yes-instance at our hand, and we output a constant size yes-instance of d -COC/dist-to- \mathcal{C} . Otherwise, we will output a graph G and a positive integer k' such that G has a d -coc set of size at most k' if and only if the input instance of α -HITTING SET is a yes-instance.

The creation of the graph G is covered next. For each $F \in \mathcal{F}$, add $k + \alpha \cdot (d-1) + 1$ copies of the graph H to G , and denote these copies as $H_F^1, \dots, H_F^{k + \alpha \cdot (d-1) + 1}$. Since each copy of H has at least one minimal d -blocking set of size α , for graph H_F^i denote an arbitrary such d -blocking set as B_F^i . Next, add a vertex v_u to the graph for each $u \in \mathcal{U}$. These vertices are called the *universe vertices*, and we denote all of them as $V_{\mathcal{U}}$, that is, $V_{\mathcal{U}} := \{v_u \mid u \in \mathcal{U}\}$. We also define $V_F := \{v_u \mid u \in F\}$ for all $F \in \mathcal{F}$.

Now, select an arbitrary bijective function $f_{F,i} : V_F \rightarrow B_F^i$ for all $F \in \mathcal{F}$ and $i \in [1, k + \alpha \cdot (d-1) + 1]$. Next, add an edge between vertex v_u and vertex $f_{F,i}(v_u)$ for all $F \in \mathcal{F}$, $u \in F$ and $i \in [1, k + \alpha \cdot (d-1) + 1]$. In other words, we associate each vertex of d -blocking set B_F^i with a different universe vertex corresponding to an element in F , and add an edge between the associated vertices. This concludes the construction of the graph G .

Observe that $G - V_{\mathcal{U}}$ is a disjoint union of graphs of \mathcal{C} , and as \mathcal{C} is closed under taking the disjoint union, $V_{\mathcal{U}}$ is a modulator to \mathcal{C} . Our output instance of d -COC/dist-to- \mathcal{C} is then $(G, k', V_{\mathcal{U}})$, where $k' := k + (k + \alpha \cdot (d-1) + 1) \cdot |\mathcal{F}| \cdot \text{OPT}$.

First, note that the parameter did not change as $|V_{\mathcal{U}}| = |\mathcal{U}|$. Next, we show that the two instances are also equivalent.

▷ **Claim.** If $(\mathcal{U}, \mathcal{F}, k)$ is a yes-instance of α -HITTING SET, then $(G, k', V_{\mathcal{U}})$ is a yes-instance of d -COC/dist-to- \mathcal{C} .

Proof. Let $X \subseteq \mathcal{U}$ be a set such that $|X| \leq k$ and the intersection of X and each set of \mathcal{F} is non-empty.

We construct a solution S for the d -COC/dist-to- \mathcal{C} instance $(G, k', V_{\mathcal{U}})$ as follows. First, we construct a subset S' of the solution. For each $u \in X$, put universe vertex v_u in the set S' , for each $u \in \mathcal{U} \setminus X$, put all vertices in $N_G(v_u)$ in the set S' . Begin with an empty set S . Then, for each $F \in \mathcal{F}$ and $i \in [1, k + \alpha \cdot (d - 1) + 1]$, add a minimum cardinality d -coc set of the graph $H_F^i - S'$ to S . Finally, also add the vertices of S' to S .

The set S contains at most k vertices of $V_{\mathcal{U}}$. Moreover, S is a d -coc set, as each vertex of $V_{\mathcal{U}}$ is either in S or all its neighbours are in S , and moreover, we choose vertices corresponding to a solution within each copy of H . All that remains is arguing that for an arbitrary $F \in \mathcal{F}$ and $i \in [1, k + \alpha \cdot (d - 1) + 1]$, S selects at most OPT vertices of H_F^i . By the fact that $X \cap F$ is non-empty, we see that $S' \cap V(H_F^i) \subsetneq B_F^i$. Moreover, the graph $H_F^i - S'$ is not connected to any vertex of $V_{\mathcal{U}}$, nor of another copy of the graph H . By the fact that B_F^i is a minimal d -blocking set, and that S' does not contain all vertices of it, we see that S selects OPT vertices of H_F^i . Hence, $|S| \leq k + (k + \alpha \cdot (d - 1) + 1) \cdot |\mathcal{F}| \cdot \text{OPT} = k'$, and we indeed have a yes-instance of d -COC/dist-to- \mathcal{C} . ◀

▷ **Claim.** If $(G, k', V_{\mathcal{U}})$ is a yes-instance of d -COC/dist-to- \mathcal{C} , then $(\mathcal{U}, \mathcal{F}, k)$ is a yes-instance of α -HITTING SET.

Proof. Let S be a d -coc set of G of size at most k' . Define $X := \{u \in \mathcal{U} \mid v_u \in V_{\mathcal{U}} \cap S\}$, that is, X is a subset of the universe corresponding to the universe vertices selected by S . Next, we show that X has a non-empty intersection with all $F \in \mathcal{F}$, and $|X| \leq k$.

For the size-bound, we know that any d -coc set of G must select at least OPT vertices of each copy of the graph H . Since there are exactly $(k + \alpha \cdot (d - 1) + 1) \cdot |\mathcal{F}|$ copies of H in G , at most k budget is left for the selection of vertices of $V_{\mathcal{U}}$.

Now, assume that for some $F \in \mathcal{F}$, we have $F \cap X = \emptyset$. Consider an arbitrary $u \in F$. We know that vertex v_u is not in S . However, v_u is adjacent to exactly one vertex of graph H_F^i for all $i \in [1, k + \alpha \cdot (d - 1) + 1]$. Given that v_u cannot be in a connected component of size more than d in $G - S$, we know that for at most $d - 1$ graphs of $H_F^1, \dots, H_F^{k + \alpha \cdot (d - 1) + 1}$ it can be the case that the neighbour of v_u in the graph is not in S . Now, since this is true for all α vertices in V_F , we see that overall, for at most $\alpha \cdot (d - 1)$ of the graphs $H_F^1, \dots, H_F^{k + \alpha \cdot (d - 1) + 1}$ their respective minimal d -blocking set is not fully contained in S . Hence, there are at least $k + 1$ copies of H for which a d -blocking set is contained in S . This in turn implies that the solution S selects at least OPT + 1 vertices of $k + 1$ copies of H , and we know that at least OPT vertices of any other copy must be selected. We see that the size of S is now at least $(k + \alpha \cdot (d - 1) + 1) \cdot |\mathcal{F}| \cdot \text{OPT} + k + 1$, which contradicts the upper bound on the size of S . ◀

By the claims above, the input and output instances are equivalent. Moreover, the parameter did not change, and the runtime of the reduction is polynomial in the input size. Hence, it is a linear parameter transformation, and the lower bound of α -HITTING SET is transferred to d -COC/dist-to- \mathcal{C} . ◀

It should be noted that the main difference between our construction and the one by Hols et al. [26] is that we use more copies of the graph H to account for the fact that d is no longer fixed to be 1, like it was for the VERTEX COVER problem.

4 Subdivided comb graphs with arbitrarily large minimal d -blocking sets

In this section we prove our main result Theorem 1, that is, we study the d -COC problem when the parameter is the distance to subdivided comb graphs. We denote this problem by d -COC/dist-to-subdivided-combs. We define the d -COC/dist-to-subdivided-combs problem formally below, and restate Theorem 1.

<p>d-COC/dist-to-subdivided-combs Input: Graph G, $M \subseteq V(G)$ such that $G - M$ is a disjoint union of subdivided comb graphs, integer $k \geq 0$ Parameter: M Question: Does there exist a d-coc set S of G of size at most k?</p>
--

► **Theorem 1.** *For any integer $d \geq 2$, d -COMPONENT ORDER CONNECTIVITY does not admit a polynomial compression when the parameter is distance to subdivided comb graphs, unless $\text{NP} \subseteq \text{coNP/poly}$.*

The proof of Theorem 1 is obtained by showing that the class of disjoint unions of subdivided comb graphs has unbounded minimal d -blocking set size for all integers $d \geq 2$, and hence, by Theorem 4, no polynomial compression exists for this problem. Formally, we prove Lemma 5, that together with Theorem 4 proves Theorem 1.

► **Lemma 5.** *For any integers $N \geq 1$ and $d \geq 2$, if \mathcal{C} contains all subdivided comb graphs with hairlength at most $d + 1$, then there exists a graph in \mathcal{C} that has a minimal d -blocking set of size N .*

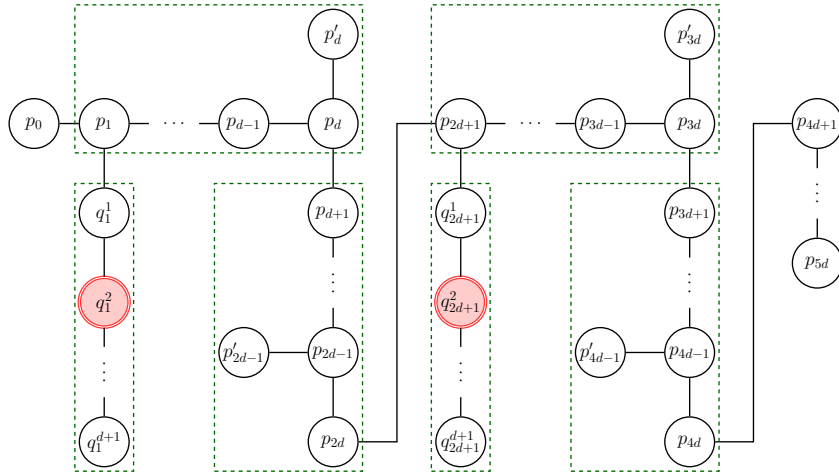
The rest of the section is devoted to the proof of Lemma 5.

The blocking set graph. For all integers $N \geq 1$, $d \geq 2$, we define a subdivided comb graph $T_{N,d}$ with hairlength at most $d + 1$, which (as we will show) has a minimal d -blocking set of size N . We call these graphs blocking set graphs.

► **Definition 6** (blocking set graph $T_{N,d}$). *For any integers $N \geq 1$, $d \geq 2$ we define $T_{N,d}$ to be the graph constructed as follows.*

- *It contains a path $P := p_0, p_1, \dots, p_{(2N+1) \cdot d}$ of length $(2N + 1) \cdot d + 1$ which is called the spine.*
- *For each $i \in \{0, 2, \dots, 2N - 2\}$, there is a path $Q_{i \cdot d + 1} := q_{i \cdot d + 1}^1, q_{i \cdot d + 1}^2, \dots, q_{i \cdot d + 1}^{d+1}$, called the $(i \cdot d + 1)$ -th leg. The vertices $q_{i \cdot d + 1}^1$ and $q_{i \cdot d + 1}^2$ are called the first and second vertex of the $(i \cdot d + 1)$ -th leg, respectively. Note that there are N legs in $T_{N,d}$.*
- *For each $i \in \{0, 2, \dots, 2N - 2\}$, add an edge between $q_{i \cdot d + 1}^1$ and $p_{i \cdot d + 1}$, that is between the first vertex of the $(i \cdot d + 1)$ -th leg and the vertex of the spine with index $i \cdot d + 1$.*
- *For each $i \in \{1, 3, \dots, 2N - 1\}$, add a pendant vertex $p'_{i \cdot d}$ to the vertex $p_{i \cdot d}$.*
- *For each $i \in \{2, 4, \dots, 2N\}$, add a pendant vertex $p'_{i \cdot d - 1}$ to the vertex $p_{i \cdot d - 1}$.*

Refer to Figure 1 for a sketch of the graph $T_{N,d}$. Observe that $T_{N,d}$ is a subdivided comb graph with hairlength at most $d + 1$. Next, we analyse some important properties of these graphs.



■ **Figure 1** A depiction of the graph $T_{N,d}$ when $N = 2$. The green dashed rectangles mark $3N$ vertex-disjoint connected subgraphs containing $d + 1$ vertices each as in Lemma 7. The vertices in $Y_{N,d}$, as in Definition 9, are drawn as red double circles.

Lower bound on the size of d -coc sets of $T_{N,d}$. We begin by proving a lower bound on the size of d -coc sets of $T_{N,d}$ by providing a set of vertex-disjoint connected subgraphs of $T_{N,d}$, each containing $d + 1$ vertices.

► **Lemma 7.** Fix any integers $N \geq 1, d \geq 2$, and define

- $A := \{T_{N,d}[Z] \mid Z = \{q_{i,d+1}^1, \dots, q_{i,d+1}^{d+1}\}, i \in \{0, 2, \dots, 2N - 2\}\},$
- $B := \{T_{N,d}[Z] \mid Z = \{p_{i,d+1}, \dots, p_{(i+1),d}, p'_{(i+1),d}\}, i \in \{0, 2, \dots, 2N - 2\}\},$ and
- $C := \{T_{N,d}[Z] \mid Z = \{p_{i,d+1}, \dots, p_{(i+1),d}, p'_{(i+1),d-1}\}, i \in \{1, 3, \dots, 2N - 1\}\}.$

Then, $\mathcal{D} := A \cup B \cup C$ is a collection of $3N$ vertex-disjoint connected subgraphs of $T_{N,d}$, each containing $d + 1$ vertices.

Proof. This directly follows from the description of $T_{N,d}$ in Definition 6. The described subgraphs are marked in Figure 1 for the case $N = 2$. ◀

As any d -coc set of $T_{N,d}$ must contain at least one vertex of each graph of \mathcal{D} we obtain the following lower bound.

► **Corollary 8.** For any integers $N \geq 1, d \geq 2$, any d -coc set S of $T_{N,d}$ fulfils $|S| \geq 3N$.

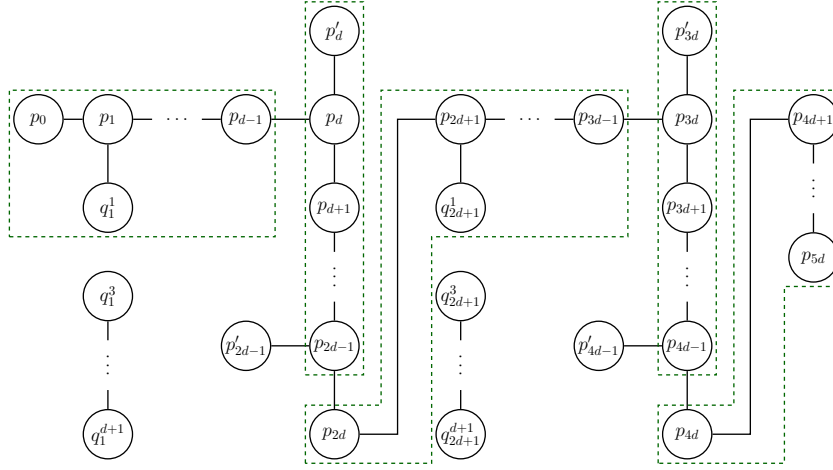
Next, for each considered graph $T_{N,d}$, we define a set $Y_{N,d}$, which we will eventually prove to be a minimal d -blocking set.

► **Definition 9.** Let N, d be integers with $d \geq 2$ and $N \geq 1$. Then, we define the vertex set $Y_{N,d} \subseteq V(T_{N,d})$ as $Y_{N,d} := \{q_{i,d+1}^2 \mid i \in \{0, 2, \dots, 2N - 2\}\}$, that is, $Y_{N,d}$ is the set consisting of the second vertices of all legs of $T_{N,d}$.

The vertices in the set $Y_{N,d}$ are marked in Figure 1 for the case $N = 2$.

No minimum solution contains all of $Y_{N,d}$. Next, we prove that any d -coc set of $T_{N,d}$ that contains all vertices of $Y_{N,d}$ has size at least $3N + 1$. As we will later see that a minimum solution has size exactly $3N$, this will end up proving that $Y_{N,d}$ is a d -blocking set.

► **Lemma 10.** For any integers $N \geq 1, d \geq 2$ consider the blocking set graph $T_{N,d}$. Then, any d -coc set S of $T_{N,d}$ such that $Y_{N,d} \subseteq S$ has cardinality at least $3N + 1$.



■ **Figure 2** A depiction of the graph $T' = T_{N,d} - Y_{N,d}$, as in the proof of Lemma 10, when $N = 2$. The green dashed rectangles indicate the subgraphs of T' that lead to the lower bound of $2N + 1$.

Proof. Let $N \geq 1, d \geq 2$ be integers and set $T := T_{N,d}$ and $Y := Y_{N,d}$. We prove that any d -coc set that contains all vertices of Y has size at least $3N + 1$.

Set $T' := T - Y$. We prove the lower bound on the size of S by providing $2N + 1$ vertex disjoint connected subgraphs of T' of size $d + 1$ each. We define

$$C_i := \begin{cases} \{p_{i \cdot d}, \dots, p_{(i+1) \cdot d-1}, q_{i \cdot d+1}^1\} & \text{if } i \in \{0, 2, \dots, 2N - 2\}, \\ \{p_{i \cdot d}, \dots, p_{(i+1) \cdot d-1}, p'_{i \cdot d}\} & \text{if } i \in \{1, 3, \dots, 2N - 1\}, \\ \{p_{2N \cdot d}, \dots, p_{(2N+1) \cdot d}\} & \text{if } i = 2N, \end{cases}$$

and $G_i := T'[C_i]$, for all $i \in [0, 2N]$. Figure 2 depicts these subgraphs for the case $N = 2$. Observe that $C_i \cap C_j = \emptyset$ for $i, j \in [0, 2N], i \neq j$, and that G_i is a connected graph for all $i \in [0, 2N]$. Furthermore, $|C_i| = d + 1$ for all $i \in [0, 2N]$, proving that any d -coc set of T' has size at least $2N + 1$. Together with the fact that $|Y| = N$, this shows that any d -coc set S of T such that $Y \subseteq S$ fulfils $|S| \geq 3N + 1$. ◀

How much of $Y_{N,d}$ can be in a minimum solution? Now, we show that even though selecting vertices of $Y_{N,d}$ is intuitively bad for the solution size, for each strict subset Y' of $Y_{N,d}$ there is a d -coc set of $T_{N,d}$ of size $3N$ that contains all vertices of Y' . This shows that a minimum d -coc set of $T_{N,d}$ has size exactly $3N$ (when combined with Corollary 8), and moreover, that $Y_{N,d}$ is a minimal d -blocking set when considering Lemma 10.

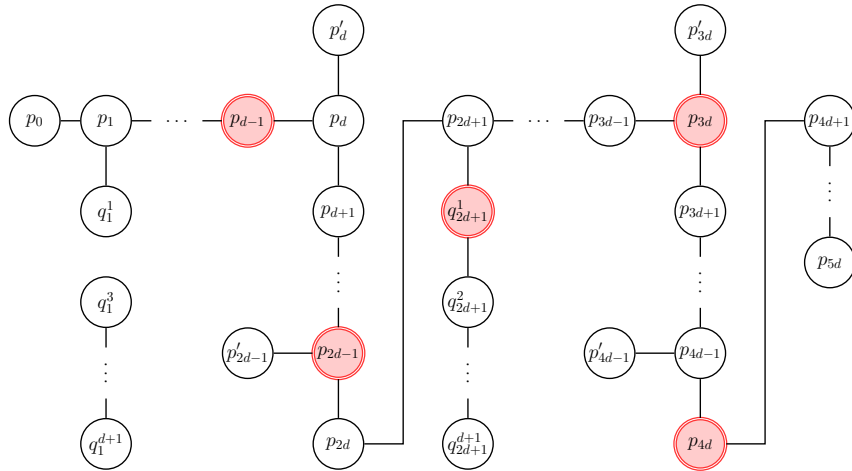
► **Lemma 11.** *For any integers $N \geq 1, d \geq 2$ consider the blocking set graph $T_{N,d}$. For any $Y' \subsetneq Y_{N,d}$, there exists a d -coc set S of $T_{N,d} - Y'$ such that $|S| = 3N - |Y'|$.*

Proof. Let $N \geq 1, d \geq 2$ be integers, set $T := T_{N,d}$, and let P be the spine of T , and $Y := Y_{N,d}$. Let Y' be an arbitrary strict subset of Y , that is $Y' \subsetneq Y$.

We show that there exists a d -coc set S of $T' := T - Y'$ of size $3N - |Y'|$. Set

$$I := \{i \cdot d + 1 \mid q_{i \cdot d+1}^2 \notin Y', i \in \{0, 2, \dots, 2N - 2\}\} \quad \text{and} \quad t := \min(I).$$

That is, I is the set of subscript indices of the legs for which the second vertex of the leg is not in Y' , and t the minimum element of the index set I . Since $Y \neq Y'$, set I is non-empty.



■ **Figure 3** A depiction of $T_{N,d} - Y'$ when $N = 2$ and $Y' = \{q_1^2\}$. The vertices drawn as red double circles form a d -coc set of size $3N - |Y'|$. Note that I, t from the proof of Lemma 11 fulfil $I = \{2d + 1\}$ and $t = 2d + 1$.

We can then construct a d -coc set S of T' of size $3N - |Y'|$, by defining

$$S := \{q_i^1 \mid i \in I\} \cup \{p_{i-d-1} \mid i \in [1, 2N], i \cdot d < t\} \cup \{p_{i-d} \mid i \in [1, 2N], i \cdot d \geq t\}.$$

Clearly, $|S| = |I| + 2N = 3N - |Y'|$. We depict solution S for the set $Y' = \{q_1^2\}$ and the case $N = 2$ in Figure 3.

Next, we argue why every connected component of $\hat{T} := T_{N,d} - (Y' \cup S)$ has size at most d . In the following, whenever we talk about connected components, we mean connected components of \hat{T} . Either the first or second vertex of every leg is in $Y' \cup S$. Thus, at most one vertex of any leg can be in a connected component together with vertices of the spine, and all other vertices of the leg are either in $Y' \cup S$, or in a connected component of size at most d together with other vertices of the same leg.

The only remaining concern is that there could be connected components of size more than d that contain vertices of the spine. Before arguing why that is not the case, we need to make some further observations.

There are two different types of pendants in the graph $T_{N,d}$, the pendants of the *first type* are $\{p'_{i-d} \mid i \in \{1, 3, \dots, 2N - 1\}\}$, and the pendants of the *second type* are the vertices $\{p'_{i-d-1} \mid i \in \{2, 4, \dots, 2N\}\}$. As either the first or the second vertex of each leg is part of $Y' \cup S$, we know that the only vertices of the legs that could be in connected components together with vertices of the spine are the first vertices. Let $X := \{q_{i-d+1}^1 \mid i \in \{0, 2, \dots, 2N - 2\}\}$ be the set of first vertices of the legs of $T_{N,d}$. The pendants of the first type, pendants of the second type, and vertices in X are all adjacent to the vertex of the spine that has the same (subscript) index they have. We additionally have that $t = j \cdot d + 1$ for some $j \in \{0, 2, \dots, 2N - 2\}$, as t is the index of some leg.

Our solution selects either vertex p_{i-d-1} or vertex p_{i-d} for any $i \in [1, 2N]$. Moreover, the vertex p_{2N-d} is guaranteed to be in S . As an immediate consequence, no connected component can contain more than d vertices of the spine. We will now show that all pendants and vertices of X are separated in \hat{T} . That is, any path in T' between two distinct vertices that are either pendants or in X contains a vertex that is in the set S . We will call such a path broken up, as it does not exist in \hat{T} .

Consider a vertex v with subscript index i that is a pendant of the first type, a pendant of the second type, or in X . Observe that, when the path from v to the closest pendant of the first type with an index lower than i , and the path from v to the closest pendant of the first type with an index higher than i are broken up by vertices of the spine, so are all other paths from v to (different) pendants of the first type. Analogous properties hold for pendants of the second type and vertices in X . This limits for which paths we need to show that they are broken up.

Pendants of the first type. Consider an arbitrary pendant $p'_{i \cdot d}$ of the first type, where $i \in \{1, 3, \dots, 2N - 1\}$. If $p_{i \cdot d} \in S$, the pendant is cut off from the rest of the graph in \hat{T} .

Otherwise, $p_{i \cdot d - 1} \in S$ and $i \cdot d < t = j \cdot d + 1$. The path from this pendant to any other pendant of the first type contains at least $2d + 1$ vertices of the spine, and is thus broken up by the solution.

The closest pendant of the second type with a larger index is $p'_{(i+1) \cdot d - 1}$. Now, we know that i is in the set $\{1, 3, \dots, 2N - 1\}$, whereas j is in the set $\{0, 2, \dots, 2N - 2\}$, which yields $i \neq j$. Since we have $i \cdot d < j \cdot d + 1$, it follows that $i < j$. This implies $(i + 1) \cdot d \leq j \cdot d$, and we obtain $(i + 1) \cdot d < j \cdot d + 1$ overall. Hence, we also select vertex $p_{(i+1) \cdot d - 1}$ into the solution, which isolates the vertex of the second type. The closest pendant of the second type with a smaller index is $p'_{(i-1) \cdot d - 1}$.³ We have $(i \cdot d) - ((i - 1) \cdot d - 1) = d + 1$, and thus, the path to this vertex contains at least $d + 2$ vertices of the spine, and is broken up.

Finally, we must argue that also paths to vertices in X are broken up. The closest vertex of X with a higher index is vertex $q_{(i+1) \cdot d + 1}^1$. As $((i + 1) \cdot d + 1) - (i \cdot d) = d + 1$, the path to this vertex is broken up. The closest vertex of X with a lower index is vertex $q_{(i-1) \cdot d + 1}^1$. Vertex $p_{i \cdot d - 1} \in S$ breaks up the path.

Pendants of the second type. Consider an arbitrary pendant $p'_{i \cdot d - 1}$ of the second type, where $i \in \{2, 4, \dots, 2N\}$. Similarly to before, any path to another pendant of the second type contains too many vertices of the spine, and is broken up.

We have already argued that all paths between pendants of the second type and pendants of the first type are broken up.

It remains to argue that also paths to vertices of X are broken up. The closest vertex of X with a lower index is vertex $q_{(i-2) \cdot d + 1}^1$, which is sufficiently far away. The closest vertex of X with a higher index is $q_{i \cdot d + 1}^1$. As we select either $p_{i \cdot d}$ or $p_{i \cdot d + 1}$, the path from $p'_{i \cdot d - 1}$ to $q_{i \cdot d + 1}^1$ is broken up.

Vertices in X . The only remaining danger is that vertices of X could be too close to each other, however, a quick glance at the definition yields that the path between two distinct vertices of X contains at least $2d + 1$ vertices of the spine, and is broken up.

Connected components of \hat{T} . We proceed to argue about the connected components that mainly consist of vertices of the spine. We select vertices $p_{i \cdot d - 1}$ into the solution as long as $i \cdot d < j \cdot d + 1$. The last vertex that is selected due to this process is vertex $p_{j \cdot d - 1}$. We first consider the connected components of vertices with index $j \cdot d - 2$ or less. As we select every d -th vertex of the spine, each considered connected component contains at most $d - 1$ vertices of the spine. By our arguments about the paths between the different types of additional vertices, we see each component can contain at most one additional vertex, and hence, it has size at most d .

³ In this and similar cases we ignore the possibility that the vertex may not exist for the sake of brevity.

Thus far, we have covered all connected components with vertices of index $j \cdot d - 2$ or less. As the spine vertex with index $j \cdot d - 1$ is selected, the next important vertex is $p_{j \cdot d}$. Observe that vertex $p_{(j+1) \cdot d}$ is in S . Hence, vertex $p_{j \cdot d}$ is in the same connected component as $\{p_{j \cdot d}, p_{j \cdot d+1}, \dots, p_{(j+1) \cdot d-1}\}$. As that are already d vertices, it is important that the connected components contains no further vertices. By definition of the integer t , the first vertex of the leg with index t is selected, and thus not in a connected component. Moreover, there exist no pendant vertices that have an index in $[j \cdot d, (j+1) \cdot d - 1]$. Another way to see that no pendant is part of the connected component is the following. We already established that any path between a vertex in X and a pendant is broken up. As that argument did not consider whether the vertex in X is itself selected or not, it still holds. Vertex $p_{j \cdot d+1}$ is part of the connected component and a neighbour of $q_{j \cdot d+1}^1$. Any path from $q_{j \cdot d+1}^1$ to a pendant would necessarily go through vertex $p_{j \cdot d+1}$, and as any such path is broken up by the solution, no pendant can be in the same connected component as $p_{j \cdot d+1}$. It follows that the connected component is also not too large.

Next, consider connected components that contain vertices with index larger than $(j+1) \cdot d$, but lower than $2N \cdot d$. We can, similarly to before, observe that we select every d -th vertex, and hence, any connected component can contain at most $d - 1$ vertices of the spine. For the same reason as earlier, no such component can contain two additional vertices, and thus, they are not too large. Observe that the vertex with the highest index that is part of the solution is $p_{2N \cdot d}$.

Finally, consider connected components that contain vertices with index larger than $2N \cdot d$. There are no legs or pendants with an index that high, and hence, there is only a single connected component that contains the vertices $\{p_{2N \cdot d+1}, \dots, p_{(2N+1) \cdot d}\}$. ◀

Combining the results. Now, we can combine our knowledge about the graph $T_{N,d}$ to prove that $Y_{N,d}$ is a minimal d -blocking set.

► **Corollary 12.** *Let $N \geq 1$ and $d \geq 2$ be integers. Then, the set $Y_{N,d}$ is a minimal d -blocking set of $T_{N,d}$ of cardinality N .*

Proof. By Definition 9, the size of $Y_{N,d}$ is N . By Corollary 8, each d -coc set of $T_{N,d}$ has size at least $3N$. Then, Lemma 11 shows that a minimum d -coc set of $T_{N,d}$ has size exactly $3N$, and moreover, that for any $Y' \subsetneq Y_{N,d}$, there is a minimum d -coc set of $T_{N,d}$ that contains all vertices of Y' . Finally, Lemma 10 shows that no d -coc set that takes all of $Y_{N,d}$ is of minimum cardinality, concluding the proof. ◀

Corollary 12 implies Lemma 5.

5 Minimal d -blocking sets of paths have size at most two

In the previous section we have illustrated that the class of subdivided comb graphs contains graphs with arbitrarily large minimal d -blocking sets, for every integer $d \geq 2$. Contrasting this result, we show that all minimal d -blocking sets of paths have size at most two for every integer $d \geq 1$. This will follow from the lemma we prove next.⁴

► **Lemma 13.** *Let G be a graph and $X \subseteq V(G)$ be a d -blocking set of G for some integer $d \geq 1$. If there exists a subset $X' \subseteq X$ such that G_1 and G_2 are distinct connected components of $G - X'$, $V(G_1) \cap X \neq \emptyset$ and $V(G_2) \cap X \neq \emptyset$, then X is not a minimal d -blocking set.*

⁴ Similar properties for (minimal) 1-blocking sets were shown in [26].

Proof. Let G be a graph and X, X' be sets with the properties described in the statement of the lemma. Moreover, let G_1 be a connected component of $G - X'$ that contains a vertex, say $x_1 \in X$, and G_2 be a different connected component of $G - X'$ that contains a vertex, say $x_2 \in X$.

Towards a contradiction, assume that X is a minimal d -blocking set. Then, there exists a minimum d -coc set S_1 of G that contains all vertices of $X \setminus \{x_1\}$, and a minimum d -coc set S_2 of G that contains all vertices of $X \setminus \{x_2\}$. Given that $x_1, x_2 \notin X'$ and $X' \subseteq X$, we see that $X' \subseteq S_1$ and $X' \subseteq S_2$. Then, it must be the case that S_i selects vertices of G_j corresponding to a minimum d -coc set of G_j for all $i, j \in \{1, 2\}$. However, we now see that the vertex set $(S_2 \setminus V(G_2)) \cup (S_1 \cap V(G_2))$ is a d -coc set of G of minimum size that contains all vertices of X , contradicting that X is a d -blocking set. ◀

An immediate corollary of this is that we obtain a bound on the size of minimal d -blocking sets of paths.

► **Corollary 14.** *For all integers $d \geq 1$, any minimal d -blocking set of a path has cardinality at most two.*

Proof. Let $P = v_1, v_2, \dots, v_n$ be a path, and $X \subseteq V(P)$ a d -blocking set. If $|X| \geq 3$, then it suffices to set $X' := \{v_i\}$, where $v_i \in X$ and X contains vertices with higher and lower subscript index than v_i . Then, Lemma 13 shows that X is not a minimal d -blocking set. ◀

For completeness, we remark that there are paths for which all minimal d -blocking sets have size exactly two. For instance, consider the path on $d + 1$ vertices. There is no minimal d -blocking set of size one, however, any two vertices of the graph form a minimal d -blocking set of size two.

In contrast to the situation we had with the distance to subdivided comb graphs, Theorem 4 cannot be used to exclude polynomial kernels when the parameter is the size of a modulator to the graph class of linear forests, that is, forests in which every tree is a path. This indicates that there is still hope for a polynomial kernel in this setting.

6 Conclusion

The d -COC problem is a relevant generalization of VERTEX COVER, and hence, efficient preprocessing algorithms are of high interest. For the VERTEX COVER problem, many kernelization algorithms are known, and usually they can be extended to also work for d -COC. In this work we show that unlike VERTEX COVER, for any integer $d \geq 2$, d -COC does not admit a kernel parameterized by the distance to subdivided comb graphs.

Several related questions gain relevance due to the new kernelization lower bound. Is a polynomial kernel possible when the parameter is even larger or at least incomparable? Reasonable possibilities would be the size of sets M such that $G - M$ consists of a disjoint union of paths, or a disjoint union of a constant number of trees, or a single subdivided comb graph, or even a single path. Answering these questions would almost certainly require very different tools from the ones we used in this paper, as the potential source of hardness would be completely different from the one that we exploited.

An interesting insight of our lower bound proof is that we show that the size of minimal d -blocking sets of families \mathcal{C} still remains a bottleneck for getting polynomial kernels for d -COC parameterized by the size of a vertex modulator to \mathcal{C} . The contrast between VERTEX COVER and d -COC is that for the former, the class \mathcal{C} of forests has minimal blocking sets of size at most two (implicitly proved in [27]), whereas when $d \geq 2$, the size of minimal d -blocking sets is unbounded even for subdivided comb graphs.

There is a result by Bougeret, Jansen & Sau [4] that complements the sufficient condition defined by Hols, Kratsch & Pieterse [26]. In particular, Bougeret et al. show that if \mathcal{C} is a minor-closed class, then the VERTEX COVER problem admits a polynomial kernel parameterized by the size of a vertex modulator to \mathcal{C} if and only if there is a bound on the size of minimal blocking sets of graphs in \mathcal{C} . An interesting open question is whether we can obtain such a characterization also for d -COC. As of now, Theorem 4 only shows one side of this characterization when \mathcal{C} is additionally closed under taking the disjoint union. If this characterization was also possible for d -COC, then Corollary 14 would immediately imply a polynomial kernel for d -COC parameterized by a vertex modulator to a linear forest.



References

- 1 Faisal N. Abu-Khzam, Rebecca L. Collins, Michael R. Fellows, Michael A. Langston, W. Henry Suters, and Christopher T. Symons. Kernelization algorithms for the vertex cover problem: Theory and experiments. In Lars Arge, Giuseppe F. Italiano, and Robert Sedgewick, editors, *Proceedings of the Sixth Workshop on Algorithm Engineering and Experiments and the First Workshop on Analytic Algorithmics and Combinatorics, New Orleans, LA, USA, January 10, 2004*, pages 62–69. SIAM, 2004.
- 2 R. Balasubramanian, Michael R. Fellows, and Venkatesh Raman. An improved fixed-parameter algorithm for vertex cover. *Inf. Process. Lett.*, 65(3):163–168, 1998. doi:10.1016/S0020-0190(97)00213-5.
- 3 Sriram Bhyravarapu, Satyabrata Jana, Saket Saurabh, and Roohani Sharma. Difference determines the degree: Structural kernelizations of component order connectivity. In Neeldhara Misra and Magnus Wahlström, editors, *18th International Symposium on Parameterized and Exact Computation, IPEC 2023, September 6-8, 2023, Amsterdam, The Netherlands*, volume 285 of *LIPICs*, pages 5:1–5:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.IPEC.2023.5.
- 4 Marin Bougeret, Bart M. P. Jansen, and Ignasi Sau. Bridge-depth characterizes which minor-closed structural parameterizations of vertex cover admit a polynomial kernel. *SIAM J. Discret. Math.*, 36(4):2737–2773, 2022. doi:10.1137/21M1400766.
- 5 Marin Bougeret and Ignasi Sau. How much does a treedepth modulator help to obtain polynomial kernels beyond sparse graphs? *Algorithmica*, 81(10):4043–4068, 2019. doi:10.1007/S00453-018-0468-8.
- 6 Jonathan F. Buss and Judy Goldsmith. Nondeterminism within P. *SIAM J. Comput.*, 22(3):560–572, 1993. doi:10.1137/0222038.
- 7 Jianer Chen, Iyad A. Kanj, and Weijia Jia. Vertex cover: Further observations and further improvements. *J. Algorithms*, 41(2):280–301, 2001. doi:10.1006/JAGM.2001.1186.
- 8 Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theor. Comput. Sci.*, 411(40-42):3736–3756, 2010. doi:10.1016/J.TCS.2010.06.026.
- 9 Jianer Chen, Lihua Liu, and Weijia Jia. Improvement on vertex cover for low-degree graphs. *Networks*, 35(4):253–259, 2000. doi:10.1002/1097-0037(200007)35:4<253::AID-NET3>3.0.CO;2-K.
- 10 Benny Chor, Mike Fellows, and David W. Juedes. Linear kernels in linear time, or how to save k colors in $O(n^2)$ steps. In Juraj Hromkovic, Manfred Nagl, and Bernhard Westfechtel, editors, *Graph-Theoretic Concepts in Computer Science, 30th International Workshop, WG 2004, Bad Honnef, Germany, June 21-23, 2004, Revised Papers*, volume 3353 of *Lecture Notes in Computer Science*, pages 257–269. Springer, 2004. doi:10.1007/978-3-540-30559-0_22.
- 11 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000. doi:10.1007/S002249910009.
- 12 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

- 13 Frank K. H. A. Dehne, Michael R. Fellows, Frances A. Rosamond, and Peter Shaw. Greedy localization, iterative compression, modeled crown reductions: New FPT techniques, an improved algorithm for set splitting, and a novel 2k kernelization for vertex cover. In Rodney G. Downey, Michael R. Fellows, and Frank K. H. A. Dehne, editors, *Parameterized and Exact Computation, First International Workshop, IWPEC 2004, Bergen, Norway, September 14-17, 2004, Proceedings*, volume 3162 of *Lecture Notes in Computer Science*, pages 271–280. Springer, 2004. doi:10.1007/978-3-540-28639-4_24.
- 14 Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. *Journal of The Acm*, 61(4):23:1–23:27, 2014. doi:10.1145/2629620.
- 15 Huib Donkers and Bart M. P. Jansen. A turing kernelization dichotomy for structural parameterizations of F-minor-free deletion. *J. Comput. Syst. Sci.*, 119:164–182, 2021. doi:10.1016/J.JCSS.2021.02.005.
- 16 Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: basic results. *SIAM J. Comput.*, 24(4):873–921, 1995. doi:10.1137/S0097539792228228.
- 17 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.
- 18 Pål Grønås Drange, Markus S. Dregi, and Pim van 't Hof. On the computational complexity of vertex integrity and component order connectivity. *Algorithmica*, 76(4):1181–1202, 2016. doi:10.1007/S00453-016-0127-X.
- 19 Michael R. Fellows. Blow-ups, win/win's, and crown rules: Some new directions in FPT. In Hans L. Bodlaender, editor, *Graph-Theoretic Concepts in Computer Science, 29th International Workshop, WG 2003, Elspeet, The Netherlands, June 19-21, 2003, Revised Papers*, volume 2880 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2003. doi:10.1007/978-3-540-39890-5_1.
- 20 Michael R. Fellows, Lars Jaffke, Aliz Izabella Király, Frances A. Rosamond, and Mathias Weller. What is known about vertex cover kernelization? In Hans-Joachim Böckenhauer, Dennis Komm, and Walter Unger, editors, *Adventures between Lower Bounds and Higher Altitudes – Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday*, volume 11011 of *Lecture Notes in Computer Science*, pages 330–356. Springer, 2018. doi:10.1007/978-3-319-98355-4_19.
- 21 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: theory of parameterized preprocessing*. Cambridge University Press, 2019. doi:10.1017/9781107415157.
- 22 Fedor V. Fomin and Torstein J. F. Strømme. Vertex cover structural parameterization revisited. In Pinar Heggernes, editor, *Graph-Theoretic Concepts in Computer Science - 42nd International Workshop, WG 2016, Istanbul, Turkey, June 22-24, 2016, Revised Selected Papers*, volume 9941 of *Lecture Notes in Computer Science*, pages 171–182, 2016. doi:10.1007/978-3-662-53536-3_15.
- 23 Daniel Gross, Monika Heinig, Lakshmi Iswara, L William Kazmierczak, Kristi Luttrell, John T Saccoman, and Charles Suffel. A survey of component order connectivity models of graph theoretic networks. *WSEAS Transactions on Mathematics*, 12(9):895–910, 2013.
- 24 David G. Harris and N. S. Narayanaswamy. A faster algorithm for vertex cover parameterized by solution size. In Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov, editors, *41st International Symposium on Theoretical Aspects of Computer Science, STACS 2024, March 12-14, 2024, Clermont-Ferrand, France*, volume 289 of *LIPICs*, pages 40:1–40:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICs.STACS.2024.40.
- 25 Danny Hermelin and Xi Wu. Weak compositions and their applications to polynomial lower bounds for kernelization. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 104–113. SIAM, 2012. doi:10.1137/1.9781611973099.9.

- 26 Eva-Maria C. Hols, Stefan Kratsch, and Astrid Pieterse. Elimination distances, blocking sets, and kernels for vertex cover. *SIAM J. Discret. Math.*, 36(3):1955–1990, 2022. doi:10.1137/20M1335285.
- 27 Bart M. P. Jansen and Hans L. Bodlaender. Vertex cover kernelization revisited: Upper and lower bounds for a refined parameter. In Thomas Schwentick and Christoph Dürr, editors, *28th International Symposium on Theoretical Aspects of Computer Science, STACS 2011, March 10-12, 2011, Dortmund, Germany*, volume 9 of *LIPICs*, pages 177–188. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011. doi:10.4230/LIPICs.STACS.2011.177.
- 28 Bart M. P. Jansen and Hans L. Bodlaender. Vertex cover kernelization revisited - upper and lower bounds for a refined parameter. *Theory Comput. Syst.*, 53(2):263–299, 2013. doi:10.1007/S00224-012-9393-4.
- 29 Bart M. P. Jansen and Astrid Pieterse. Polynomial kernels for hitting forbidden minors under structural parameterizations. *Theor. Comput. Sci.*, 841:124–166, 2020. doi:10.1016/J.TCS.2020.07.009.
- 30 Bart MP Jansen. *The power of data reduction: Kernels for fundamental graph problems*. PhD thesis, Utrecht University, 2013.
- 31 Samir Khuller. Algorithms column: the vertex cover problem. *SIGACT News*, 33(2):31–33, 2002. doi:10.1145/564585.564598.
- 32 Mithilesh Kumar and Daniel Lokshtanov. A $2k$ kernel for l -component order connectivity. In Jiong Guo and Danny Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*, volume 63 of *LIPICs*, pages 20:1–20:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.IPEC.2016.20.
- 33 Michael Lampis. A kernel of order $2k - c \log k$ for vertex cover. *Inf. Process. Lett.*, 111(23-24):1089–1091, 2011. doi:10.1016/J.IPL.2011.09.003.
- 34 Diptapriyo Majumdar, Venkatesh Raman, and Saket Saurabh. Polynomial kernels for vertex cover parameterized by small degree modulators. *Theory of Computing Systems*, 62(8):1910–1951, 2018. doi:10.1007/S00224-018-9858-1.
- 35 N. S. Narayanaswamy, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. LP can be a cure for parameterized problems. In Christoph Dürr and Thomas Wilke, editors, *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, February 29th - March 3rd, 2012, Paris, France*, volume 14 of *LIPICs*, pages 338–349. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPICs.STACS.2012.338.
- 36 Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006. doi:10.1093/ACPROF:OSO/9780198566076.001.0001.
- 37 Rolf Niedermeier and Peter Rossmanith. Upper bounds for vertex cover further improved. In Christoph Meinel and Sophie Tison, editors, *STACS 99, 16th Annual Symposium on Theoretical Aspects of Computer Science, Trier, Germany, March 4-6, 1999, Proceedings*, volume 1563 of *Lecture Notes in Computer Science*, pages 561–570. Springer, 1999. doi:10.1007/3-540-49116-3_53.
- 38 Rolf Niedermeier and Peter Rossmanith. On efficient fixed-parameter algorithms for weighted vertex cover. *J. Algorithms*, 47(2):63–77, 2003. doi:10.1016/S0196-6774(03)00005-1.
- 39 Arezou Soleimanfallah and Anders Yeo. A kernel of order $2k - c$ for vertex cover. *Discret. Math.*, 311(10-11):892–895, 2011. doi:10.1016/J.DISC.2011.02.014.
- 40 Ulrike Stege and Michael Ralph Fellows. An improved fixed parameter tractable algorithm for vertex cover. *Technical report/Departement Informatik, ETH Zürich*, 318, 1999. doi:10.3929/ethz-a-006653305.
- 41 Mingyu Xiao. Linear kernels for separating a graph into components of bounded size. *Journal of Computer and System Sciences*, 88:260–270, 2017. doi:10.1016/J.JCSS.2017.04.004.

Dynamic Parameterized Feedback Problems in Tournaments

Anna Zych-Pawlewicz  

Institute of Informatics, University of Warsaw, Poland

Marek Żochowski 

Institute of Informatics, University of Warsaw, Poland

Abstract

In this paper we present the first dynamic algorithms for the problem of K-FEEDBACK ARC SET IN TOURNAMENTS (K-FAST) and the problem of K-FEEDBACK VERTEX SET IN TOURNAMENTS (K-FVST). Our algorithms maintain a dynamic tournament on n vertices altered by redirecting the arcs, and answer if the tournament admits a feedback arc set (or respectively feedback vertex set) of size at most K , for some chosen parameter K . For dynamic K-FAST we offer two algorithms. In the promise model, where we are guaranteed, that the size of the solution does not exceed $g(K)$ for some computable function g , we give an $O(\sqrt{g(K)})$ update and $O(3^K K \sqrt{K})$ query algorithm. In the general setting without any promise, we offer an $O(\log^2 n)$ update and $O(3^K K \log^2 n)$ query time algorithm for dynamic K-FAST. For dynamic K-FVST we offer an algorithm working in the promise model, which admits $O(g^5(K))$ update and $O(3^K K^3 g(K))$ query time.

2012 ACM Subject Classification Theory of computation \rightarrow Parameterized complexity and exact algorithms; Theory of computation \rightarrow Data structures design and analysis

Keywords and phrases dynamic algorithms, parameterized algorithms, feedback arc set, feedback vertex set, tournaments

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.22

Related Version *Full Version:* <https://arxiv.org/abs/2404.12907> [28]

Funding National Science Centre, Poland, Grant 2017/26/D/ST6/00264

1 Introduction and Related Work

In this paper we study feedback set problems in dynamic tournaments. A tournament is a directed graph where every pair of vertices is connected by exactly one arc. The feedback arc (resp. vertex) set of a given directed graph is a set of arcs (resp. vertices) whose removal makes the graph acyclic. The problems we focus on here are the FEEDBACK ARC SET IN TOURNAMENTS and the FEEDBACK VERTEX SET IN TOURNAMENTS. In the classical static setting, these problems are very well known and are defined as follows:

- K-FEEDBACK ARC SET IN TOURNAMENTS (K-FAST): Given a tournament $T = (V, E)$ and a positive integer K , does there exist a subset $F_E \subseteq E$ of at most K arcs whose removal makes T acyclic.
- K-FEEDBACK VERTEX SET IN TOURNAMENTS (K-FVST): Given a tournament $T = (V, E)$ and a positive integer K , does there exist a subset $F_V \subseteq V$ of at most K vertices whose removal makes T acyclic.

Both the above problems are flag problems in the area of parameterized complexity and textbook examples for the branching technique. Feedback arc sets in tournaments have applications in voting systems and rank aggregation, and are well studied from the combinatorial [9, 11, 25] and algorithmic [27, 16, 6, 3, 10, 12] points of view. Unfortunately, the K-FAST problem is NP-hard [2]. The fastest parameterized algorithm achieves $2^{O(\sqrt{K})} n^{O(1)}$ running time [12], where n is the size of the input tournament. In this paper, however, we



© Anna Zych-Pawlewicz and Marek Żochowski;
licensed under Creative Commons License CC-BY 4.0

19th International Symposium on Parameterized and Exact Computation (IPEC 2024).

Editors: Édouard Bonnet and Paweł Rzażewski; Article No. 22; pp. 22:1–22:15

Leibniz International Proceedings in Informatics



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

explore a textbook branching algorithm for this problem running in $3^K n^{O(1)}$ time [24]. The K-FVST problem also has many interesting applications, for instance in social choice theory where it is essential to the definition of a certain type of election winners [4]. It is also NP-hard [26] and has been studied from various angles [19, 20, 13, 22]. The fastest currently known parameterized algorithm for this problem runs in $O(1.6181^K + n^{O(1)})$ time [19].

Recently, there is a growing interest in studying parameterized problems in a dynamic setting. In this context, we typically consider an instance I of a problem of interest with an associated parameter K . The instance is dynamic, i.e., it is updated over time by problem specific updates, while for simplicity we assume that the problem parameter K does not change through the entire process. The goal is to provide a data structure, that allows for efficient updates to I , and upon a query efficiently provides the answer to the problem in question. The update/query running time may depend in any computable way on the parameter K , but it should be sublinear in terms of the size of I . The typical update/query running times in this setting are $f(K)$, $f(K)(\log n)^{O(1)}$, or sometimes even $f(K)n^{o(1)}$, where f is some computable function and n is the size of I . Since we allow f to be exponential, this setting applies to NP-hard problems as long as they are fixed-parameter tractable in the static setting.

Parameterized dynamic data structures were first systematically investigated by Iwata and Oka [15], followed by Alman et al. [1]. These works provided data structures with update times $f(K)$ or $f(K) \cdot (\log n)^{O(1)}$ for several classic problems such as VERTEX COVER, CLUSTER VERTEX DELETION, HITTING SET, FEEDBACK VERTEX SET, or LONGEST PATH. Other recent advances include data structures for maintaining various graph decompositions together with runs of dynamic programming procedures [5, 7, 8, 17, 21] and treatment of parameterized string problems from the dynamic perspective [23].

Alman et al. in their work [1] study the problem of K-FEEDBACK VERTEX SET in dynamic undirected graphs, where the graph is altered by edge additions and removals and the goal is to test if the graph has a feedback vertex set of size at most K . For this problem, Alman et al. propose a dynamic algorithm with $2^{O(K \log K)} \log^{O(1)} n$ amortized update time and $O(K)$ query time. It is worth mentioning, that while the K-FEEDBACK VERTEX SET problem is NP-hard on undirected graphs, the K-FEEDBACK ARC SET problem is polynomial in this class of graphs and can be efficiently maintained dynamically using dynamic connectivity algorithms [14]. So an interesting question is whether these two problems admit efficient dynamic algorithms in the class of directed graphs. In this regard Alman et al. [1] show lower bounds for the K-FEEDBACK VERTEX SET problem, which extend also to the K-FEEDBACK ARC SET problem. To be more precise, they show that in this case the dynamic algorithm requires $\Omega(f(K)m^\delta)$ update/query time for some fixed $\delta > 0$, assuming RO hypothesis (see [1]). Thus, a natural question is whether we can have more efficient dynamic algorithms at least in tournaments. In this paper we give positive answers to this question.

Our precise setting is the following. With regard to dynamic K-FAST, the goal is to design a data structure supporting the following operations:

- **Initialize**(T, n) - initialize the data structure with a given tournament T on n vertices
- **Reverse**(u, v) - reverse an arc in T between two vertices u and v of T
- **FindFAST**() - answer if there is a feedback arc set of size at most K in T .

With regard to dynamic K-FVST, the goal is to design a data structure supporting the analogous operations:

- **Initialize**(T, n) - initialize the data structure with a given tournament T on n vertices
- **Reverse**(u, v) - reverse an arc in T between two vertices u and v of T
- **FindFVST**() - answer if there is a feedback vertex set of size at most K in T .

■ **Table 1** The known results for dynamic K-FEEDBACK ARC SET. and dynamic K-FEEDBACK VERTEX SET for different classes of graphs.

	Undirected Graphs	Directed Graphs	Tournaments
K-FEEDBACK ARC SET	$\log^{\mathcal{O}(1)} n$ update $\log^{\mathcal{O}(1)} n$ query [14]	no $f(K)m^{\mathcal{O}(1)}$ algorithm assuming RO hypothesis [1]	full model: $\mathcal{O}(\log^2 n)$ update $\mathcal{O}(3^K K \log^2 n)$ query promise model: $\mathcal{O}(\sqrt{g(K)})$ update $\mathcal{O}(3^K K \sqrt{K})$ query
K-FEEDBACK VERTEX SET	$2^{\mathcal{O}(K \log K)} \log^{\mathcal{O}(1)} n$ amortized update $\mathcal{O}(K)$ query [1]	no $f(K)m^{\mathcal{O}(1)}$ algorithm assuming RO hypothesis [1]	promise model: $\mathcal{O}(g^5(K))$ update $\mathcal{O}(3^K K^3 g(K))$ query

The above setting is referred to as the *full* model. A popular restriction of this setting, introduced in generality by Alman et. al. [1], is called the *promise* model. The promise model applied to our setting provides the data structure with a guarantee, that the feedback arc set (or respectively feedback vertex set) remains of size bounded by $g(K)$ for some computable function g for the entire process. Some algorithms provided by Alman et al. [1] work only in this restricted setting.

Our results

For the dynamic K-FAST problem we offer two data structures. In the promise model we propose an $\mathcal{O}(\sqrt{g(K)})$ update and $\mathcal{O}(3^K K \sqrt{K})$ query data structure which does not need to know $g(K)$. In the full model, we offer an $\mathcal{O}(\log^2 n)$ update and $\mathcal{O}(3^K K \log^2 n)$ query data structure. For the dynamic K-FVST problem we offer a data structure which works in the promise model, with $\mathcal{O}(g^5(K))$ update and $\mathcal{O}(3^K K^3 g(K))$ query time. This data structure does need to know $g(K)$. Our results for dynamic feedback set problems, compared to the related results of Alman et al. [1] are shown in Table 1. All our running times are worst case (i.e., not amortized).

As a side result, we propose two dynamic data structures that can efficiently find triangles (i.e., directed cycles of length three) in dynamic tournaments. The efficiency of these data structures depends on parameter $\text{ADT}(T)$, which is the maximum number of arc-disjoint triangles in T . The first data structure admits $\mathcal{O}(\sqrt{\text{ADT}(T)})$ update time and $\mathcal{O}(\text{ADT}(T)\sqrt{\text{ADT}(T)})$ query time. The second data structure admits $\mathcal{O}(\log^2 n)$ update time and $\mathcal{O}(\text{ADT}(T) \log^2 n)$ query time. The data structures do not need to know $\text{ADT}(T)$. We believe that the triangle detection data structures may be of independent interest.

In the next sections we provide an illustrative overview of our techniques and ideas used to obtain our results, with the main focus on the dynamic K-FAST problem in the promise model. The rigorous proofs, as well as details for the dynamic K-FAST problem in the full model and dynamic K-FVST problem in the promise model are moved to the full version [28] due to space limitations.

2 Triangle Detection Data Structures

For both the dynamic K-FAST and the dynamic K-FVST problem, in order to answer the query efficiently, our plan is to run the standard (static) branching algorithms (see Algorithm 1 and Algorithm 2 respectively) upon every query. The branching algorithm for K-FAST relies

on the folklore knowledge that the minimum feedback arc set in a tournament is equivalent to the minimum set of arcs whose reversal makes the tournament acyclic. It is also a folklore fact that a tournament has a cycle if and only if it has a cycle of length three (that we refer to as a triangle). These folklore facts together with proofs can be found in [24]. For both K-FAST and K-FVST the branching algorithm executes at most 3^K recursive calls. In each recursive call the algorithm finds a triangle in the tournament and tries to reverse each of its edges (for the K-FAST problem) or remove each of its vertices (for the K-FVST problem). Thus, to implement the K-FAST query efficiently in the dynamic setting, we need a data structure that can quickly find a triangle in a dynamic tournament altered by arc reversals. For the dynamic K-FVST problem, the data structure also needs to allow removing a limited number of vertices.

Hence, the basis for our algorithms are the triangle detection data structures, that might be of independent interest. The data structures we provide are stated independently of the feedback set problems. They rely on a different parameter, which is the maximum number of arc-disjoint triangles in the tournament. Nevertheless, our later results rely on a close connection between the maximum number of arc-disjoint triangles and the minimum feedback arc set of a tournament. Throughout the paper, for a given tournament T , we denote by $\text{FAST}(T)$ the size of the minimum feedback arc set in T , by $\text{ADT}(T)$ the maximum number of arc-disjoint triangles in T , and by $\text{FVST}(T)$ the size of the minimum feedback vertex set in T . The following fact holds.

► **Fact 1.** $\text{ADT}(T) \leq \text{FAST}(T) \leq 6(\text{ADT}(T) + 1)$.

Proof. It is easy to see that $\text{ADT}(T) \leq \text{FAST}(T)$, as in each of the $\text{ADT}(T)$ arc-disjoint triangles one arc must be taken to the feedback arc set of T . The proof of the second inequality can be found in [18] (Theorem 4). ◀

The triangle detection data structures maintain the dynamic tournament altered by reversing arcs, and allow queries for a triangle in the maintained tournament. The first data structure is given by Theorem 2 below. It is later used for dynamic K-FAST in the promise model.

► **Theorem 2** (Theorem 15 in Appendix A in [28]). *For any integer $n \in \mathbb{N}$ there exists a data structure $\mathbb{DTP}[n]$, that maintains a dynamically changing tournament T on n vertices¹ by supporting the following operations:*

1. **Initialize**(T, n) – initializes the data structure with a given tournament T on n vertices, in time $\mathcal{O}(n^2)$
2. **Reverse**(v, u) – reverses an arc between vertices v and u in T , in $\mathcal{O}(\sqrt{\text{ADT}(T)})$ time
3. **FindTriangle**() – returns a triangle from T or reports that there are no triangles, in time $\mathcal{O}(\text{ADT}(T)\sqrt{\text{ADT}(T)})$

We note here that the above data structure does not need to know the value of $\text{ADT}(T)$. The same holds for the second data structure presented next. The second triangle detection data structure gets rid of the dependence on the parameter in the update operation. This later allows us to use it for dynamic K-FAST in the full model at the cost of introducing factors poly-logarithmic in the size of the tournament.

► **Theorem 3** (Theorem 16 in Appendix A in [28]). *For any integer $n \in \mathbb{N}$ there exists a data structure $\mathbb{DT}[n]$, that maintains a dynamically changing tournament T on n vertices¹ by supporting the following operations:*

1. **Initialize**(T, n) – initializes the data structure with a given tournament T on n vertices, in time $\mathcal{O}(n^2)$
2. **Reverse**(v, u) – reverses an arc between vertices v and u in T , in time $\mathcal{O}(\log^2 n)$
3. **FindTriangle**() – returns a triangle from T or reports that there are no triangles, in time $\mathcal{O}(\text{ADT}(T) \log^2 n)$

Both data structures are described in detail in Appendix A in [28]. In this overview, we mainly focus on describing the first data structure $\mathbb{DTTP}[n]$ of Theorem 2, which also sheds some light on the data structure $\mathbb{DT}[n]$ of Theorem 3, as both data structures are based on similar main ideas.

In particular, both data structures maintain the same basic information related to the dynamic tournament T . First and foremost, both data structures maintain n sets called *indegree buckets*, which partition the vertices of T according to their indegrees in T . The indegree bucket $\text{Db}_T[d]$ stores vertices of indegree d . The indegree buckets alone let us easily determine if the tournament is acyclic due to the following fact (which can be found for instance in [10]).

► **Fact 4** (Fact 19 in Appendix A.1 in [28]). *A tournament is acyclic if and only if all its indegree buckets have size one.*

By $\max_{\text{Db}}(T)$ we denote the maximum size of an indegree bucket for the maintained tournament T . It is easy to see, that $\max_{\text{Db}}(T) \leq 2\text{FAST}(T) + 1 \in \mathcal{O}(\text{ADT}(T))$, because one arc reversal can remove at most two vertices from the indegree bucket of maximum size, and we have to remove all but one to make the tournament acyclic. With a bit more care one can show the following bound.

► **Fact 5** (Lemma 25 in Appendix A.1 in [28]). $\max_{\text{Db}}(T) \leq 8\sqrt{\text{ADT}(T) + 1} + 8$.

Both data structures also maintain a set **Empty** of indices of indegree buckets that are empty, i.e., $\text{Empty} = \{d \in \{0, \dots, n-1\} : \text{Db}_T[d] = \emptyset\}$. Since each arc reversal can place up to two vertices in the empty buckets, and since by Fact 4 there are no empty buckets after reversing the arcs of a minimum feedback arc set, the following bound holds.

► **Fact 6** (Lemma 27 in Appendix A.1 in [28]). $|\text{Empty}| \leq 2\text{FAST}(T) \in \mathcal{O}(\text{ADT}(T))$

Both the indegree buckets and the set **Empty** are straightforward to maintain in a constant time per arc reversal, since every arc reversal affects the indegrees of exactly two vertices (see Lemma 35 in [28] for the details). The data structures also rely on some other information that is straightforward to maintain, such as adjacency matrix of the tournament and similar basic structures. These are not crucial enough to be mentioned in this short description, but they are detailed in Appendix A in [28] instead.

The promise data structure $\mathbb{DTTP}[n]$ additionally maintains a set **Back** of *back arcs*. An arc uv is called a back arc if $d_T(u) \geq d_T(v)$, where $d_T(x)$ stands for the indegree of x in T . The back arcs are another natural obstacle to T being acyclic, as every back arc belongs to some triangle in T (see Lemma 21 in Appendix A in [28]). Reversing the arcs of a minimum feedback arc set of T gets rid of all cycles and thus also gets rid of all back arcs in T .

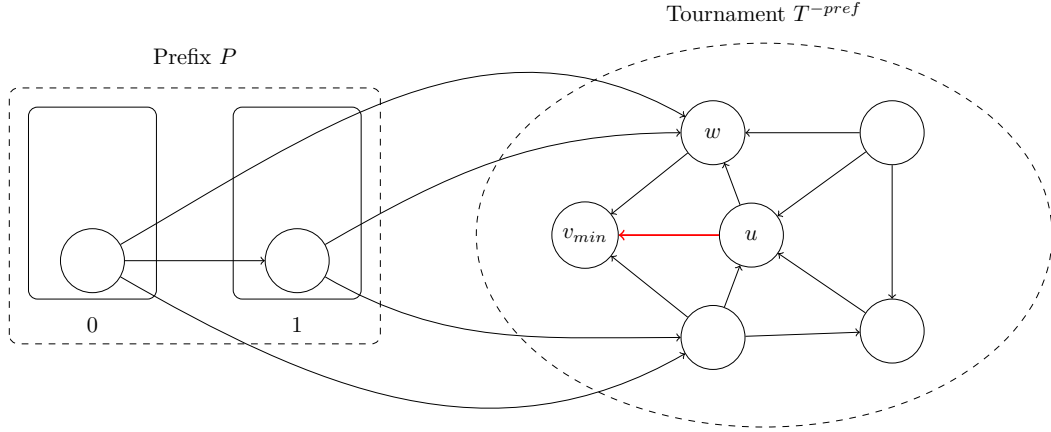
The set **Back** changes its size by at most $\mathcal{O}(\sqrt{\text{ADT}(T)})$ upon arc reversal for the following reason. Consider reversing arc uv . Any arc that after the reversal becomes a back arc or stops being a back arc has one endpoint in either u or v . The candidates for the other endpoint of

¹ All our data structures assume that the vertices of T are indexed with numbers from 0 to $n-1$.

such arc have indegrees differing by at most one from $d_T(u)$ or $d_T(v)$ (see Lemma 31 in [28] for more details). This gives (by Fact 5) $O(\sqrt{\text{ADT}(T)})$ candidate arcs for altering the set **Back**. This observation has two important consequences. Firstly, maintaining the set of back arcs **Back** takes $O(\sqrt{\text{ADT}(T)})$ time per update. Secondly, we get the bound for the size of the set of back arcs **Back**.

► **Fact 7** (Lemma 33 in Appendix A.1 in [28]). $|\text{Back}| \in O(\text{ADT}(T)\sqrt{\text{ADT}(T)})$

Fact 7 follows for the following reason. Reversing $\text{FAST}(T)$ arcs gets rid of all back arcs, and one arc reversal gets rid of at most $O(\sqrt{\text{ADT}(T)})$ back arcs. Thus, $|\text{Back}| \in O(\text{FAST}(T)\sqrt{\text{ADT}(T)})$. By Fact 1 the bound of Fact 7 follows.



■ **Figure 1** Tournament T , its prefix P and tournament $T^{-\text{pref}}$. Arcs between all pairs of vertices are present in T , however some arcs are not drawn for the sake of readability.

We now move on to describing how our data structures detect triangles. To support this operation, we first define the *prefix* P of a tournament T as the set of vertices in the maximum prefix of indegree buckets of size one (see Figure 1 for illustration and Definition 22 in [28] for a more formal definition). The subtournament $T[P]$ of T induced by its prefix P is acyclic by Fact 4. Since we cannot find triangles inside the prefix P , we are more interested in tournament $T^{-\text{pref}} = T[V(T) \setminus P]$ which stands for tournament T induced by all its vertices excluding prefix. Both the prefix P of a tournament T and the remaining tournament $T^{-\text{pref}}$ are illustrated in Figure 1.

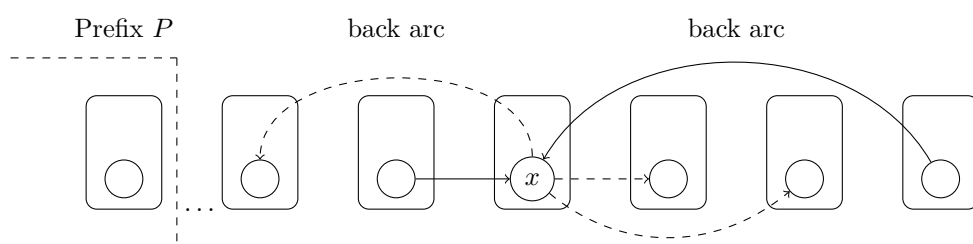
The data structures $\mathbb{DTTP}[n]$ and $\mathbb{DT}[n]$ follow the same general approach. In order to find a triangle, they first search for a vertex v_{\min} of minimum indegree in $T^{-\text{pref}}$ (see Figure 1). One can find the vertex v_{\min} in $O(|\text{Empty}|) = O(\text{ADT}(T))$ time by iterating through the set **Empty**. Clearly, the indegree buckets whose indices range from 0 to $|P| - 1$ are not empty, by the definition of prefix P . Since we want to skip vertices of P , we are interested in the first index larger than $|P| - 1$ which is not in **Empty**. This is the index of the bucket containing v_{\min} . We have the following bound on $d_{T^{-\text{pref}}}(v_{\min})$, which is the minimum indegree in $T^{-\text{pref}}$.

► **Fact 8** (Fact 28 in Appendix A.1 in [28]). $d_{T^{-\text{pref}}}(v_{\min}) \leq |\text{Empty}|$.

Fact 8 follows, because in $T^{-\text{pref}}$ all indegree buckets $\text{Db}_{T^{-\text{pref}}}[d]$ for $d < d_{T^{-\text{pref}}}(v_{\min})$ are empty. The number of empty indegree buckets of $T^{-\text{pref}}$ is the same as the number of empty indegree buckets of T , because $d_{T^{-\text{pref}}}(w) = d_T(w) - |P|$ for any vertex w of $T^{-\text{pref}}$.

Once v_{\min} is located, the data structures find an in-neighbor u of v_{\min} in $T^{-\text{pref}}$. Next, the data structures find a set W of $d_{T^{-\text{pref}}}(v_{\min})$ in-neighbors of u in $T^{-\text{pref}}$. Since $d_{T^{-\text{pref}}}(v_{\min}) \leq d_{T^{-\text{pref}}}(u)$, vertex u has at least $d_{T^{-\text{pref}}}(v_{\min})$ in-neighbors in $T^{-\text{pref}}$. We are bound to find a triangle $uv_{\min}w$ for some $w \in W$, because if all the arcs between W and v_{\min} were directed towards v_{\min} , that would imply that the indegree of v_{\min} in $T^{-\text{pref}}$ is more than $d_{T^{-\text{pref}}}(v_{\min})$. This is illustrated in Figure 1.

Thus, in order to find a triangle, we need an *in-neighbour listing* method to list l in-neighbours in $T^{-\text{pref}}$ of a given vertex $x \in T^{-\text{pref}}$. The method responsible for finding in-neighbours is different for the two data structures. In the promise data structure $\mathbb{DTTP}[n]$, this method (provided in detail in Lemma 40 in [28]) heavily relies on the fact that we can iterate over the set of back arcs Back . The process of finding in-neighbours by $\mathbb{DTTP}[n]$ data structure is illustrated in Figure 2.



■ **Figure 2** In-neighbour listing method in the promise model.

In order to find the in-neighbours of a given vertex x , the $\mathbb{DTTP}[n]$ data structure iterates over the indegree buckets of T starting after the prefix. For each $d > |P|$ such that $d \leq d_T(x)$, we proceed as follows. If $\text{Db}_T[d]$ is empty, we charge it to $|\text{Empty}|$ and move forward. Otherwise we iterate through vertices $w \in \text{Db}_T[d]$. If w is an in-neighbour of x , then we add it to the set of found in-neighbours W , whose size is bounded by parameter l . If w is an out-neighbour of v , then xw is a back arc and we charge such situation to $|\text{Back}|$. If by the time we reach $d = d_T(v)$ the size of W is not sufficient, we iterate through the set Back to find the remaining in-neighbours of x . The whole process takes $O(|\text{Back}| + |\text{Empty}| + l)$ time (see Lemma 40 in [28] for more details). By Fact 6, Fact 7 and Fact 8 we get the desired running time of the triangle query in the promise model.

The $\mathbb{DT}[n]$ data structure takes a different approach to find in-neighbours of a given vertex, as it does not have access to the set Back . Instead, it uses balanced search trees, what implies poly logarithmic factors in the update and query times. Due to space limitations, we defer the description of the $\mathbb{DT}[n]$ data structure to Appendix A.4 in [28].

3 Dynamic K-FEEDBACK ARC SET IN TOURNAMENTS

In this section we use the triangle detection data structures for dynamic K-FAST problem. We only focus on the main ideas, while the details are presented in Appendix B in [28]. Let us consider a standard branching algorithm which verifies whether $\text{FAST}(T) \leq K$ (shown in Algorithm 1, see [24] for correctness). Algorithm 1 can be implemented using both triangle detection data structures provided in Section 2.

We first show how to obtain the dynamic K-FAST data structure in the promise model, where $\text{FAST}(T) \leq g(K)$ at all times. Then, by Fact 1, also $\text{ADT}(T) \leq g(K)$. The procedure $\text{FindFAST}(K)$ in Algorithm 1 calls itself recursively at most 3^K times, each call employs

■ **Algorithm 1** Pseudocode for FindFAST(K).

```

Algorithm : FindFAST(K)
Output    : Verify if  $\text{FAST}(T) \leq K$ 
1 if  $T$  is acyclic then
2 |   return TRUE ;
3 if  $K = 0$  then
4 |   return FALSE ;
5  $uvw \leftarrow \text{FindTriangle}()$ ;
6 for  $xy \in \{uv, vw, wu\}$  do
7 |    $\text{Reverse}(x, y)$  ;
8 |   if FindFAST( $K - 1$ ) then
9 | |    $\text{Reverse}(y, x)$ ;
10 | |   return TRUE ;
11 |    $\text{Reverse}(y, x)$ ;
12 return FALSE ;

```

a constant number of updates and queries to a triangle detection data structure. Thus, directly by Theorem 2, employing the data structure $\mathbb{DTP}[n]$ results in a dynamic K-FAST data structure with $\mathcal{O}(3^K g(K) \sqrt{g(K)})$ query time and $\mathcal{O}(\sqrt{g(K)})$ update time.

The query time can be improved to $\mathcal{O}(3^K K \sqrt{K})$ by using procedure $\text{FindTriangle}(K)$ instead of $\text{FindTriangle}()$. The $\text{FindTriangle}(K)$ procedure works analogously to the procedure $\text{FindTriangle}()$, but has an option not to return a triangle once it detects that $\text{ADT}(T) > K$. In such case by Fact 1 also $\text{FAST}(T) > K$, and the recursive call can safely return that the solution does not exist. This is very helpful, as the procedure $\text{FindTriangle}(K)$ can stop working once any of the bounds on $|\text{Empty}|$, $|\text{Back}|$ or $\text{max}_{\text{Db}}(T)$ (given by Fact 5, Fact 6 and Fact 7) that we have in terms of $\text{ADT}(T)$ does not hold in terms of K . For instance, if $\text{max}_{\text{Db}}(T) > 8\sqrt{K} + 1 + 8$, then by Fact 5 the method $\text{FindTriangle}(K)$ is allowed to terminate returning that $\text{ADT}(T) > K$. In this way we obtain a better result stated below.

► **Theorem 9** (Theorem 49 in Appendix B in [28]). *The dynamic K-FAST problem admits a data structure with initialization time $\mathcal{O}(n^2)$, worst-case update time $\mathcal{O}(\sqrt{g(K)})$ and worst-case query time $\mathcal{O}(3^K K \sqrt{K})$ under the promise that there is a computable function g , such that the maintained tournament T always has a feedback arc set of size at most $g(K)$.*

Moving on from the promise model to the full model, we can employ the data structure $\mathbb{DT}[n]$ of Theorem 3 to implement the $\text{FindFAST}(K)$ method given in Algorithm 1. There, we cannot use $\text{FindTriangle}()$ method anymore, as its running time depends on $\text{ADT}(T)$, which is not bounded as in the promise model. Instead, we again use $\text{FindTriangle}(K)$ method, which again reports $\text{ADT}(T) > K$ if the necessary bounds do not hold in terms of K . Thanks to that, $\text{FindTriangle}(K)$ runs in $\mathcal{O}(K \log^2 n)$ time. If the procedure $\text{FindTriangle}(K)$ fails to find a triangle, we can safely report that no solution exists, as this means that $\text{FAST}(T) \geq \text{ADT}(T) > K$. This way, we arrive at the following result.

► **Theorem 10** (Theorem 50 in Appendix B in [28]). *The dynamic K-FAST problem admits a data structure with initialization time $\mathcal{O}(n^2)$, worst-case update time $\mathcal{O}(\log^2 n)$ and worst-case query time $\mathcal{O}(3^K K \log^2 n)$.*

4 Dynamic K-FEEDBACK VERTEX SET IN TOURNAMENTS in the promise model

In Appendix C in [28] we present the data structure for the dynamic K-FVST problem in the promise model. The main idea is the same as for the dynamic K-FAST problem in the promise model. We want to be able to quickly find a triangle in tournament T and perform a standard static branching algorithm for K-FVST, presented in Algorithm 2.

■ **Algorithm 2** Pseudocode for $\text{FindFVST}(K)$.

Algorithm : $\text{FindFVST}(K)$
Output : Verify if $\text{FVST}(T) \leq K$

```

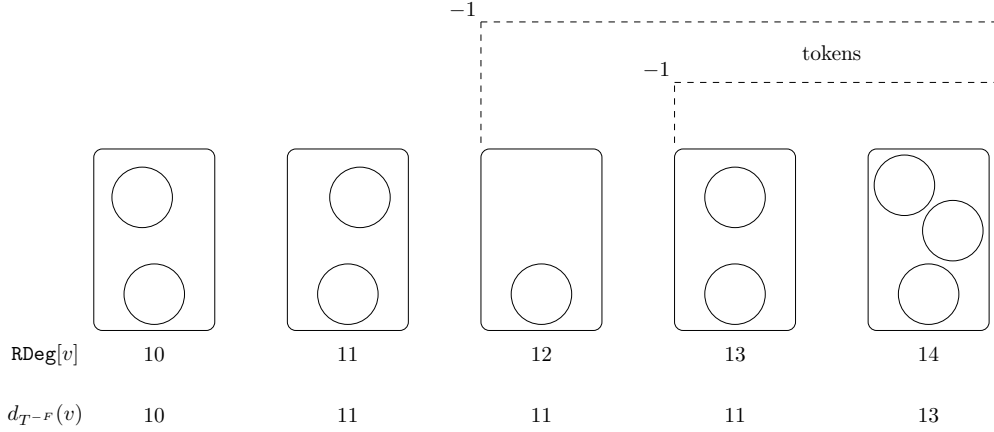
1 if  $T$  is acyclic then
2   | return TRUE ;
3 if  $K = 0$  then
4   | return FALSE ;
5  $uvw \leftarrow \text{FindTriangle}()$ ;
6 for  $x \in \{u, v, w\}$  do
7   | Remove( $x$ ) ;
8   | if  $\text{FindFVST}(K - 1)$  then
9     | Restore( $x$ );
10    | return TRUE ;
11  | Restore( $x$ );
12 return FALSE ;
```

The branching algorithm (Algorithm 2) finds a triangle in the tournament, and then branches recursively with each of the triangle vertices removed. The correctness is again due to the fact that a tournament is acyclic if and only if it has no triangles. To implement this algorithm, we not only need a method to find a triangle in the maintained tournament, but we also need to support vertex removals and restorations. These are significantly more complex than edge reversals, as they change the indegree of up to $(n - 1)$ vertices. This loss of locality poses a number of problems, including maintaining indegree buckets, maintaining the set of empty indegree buckets, or even keeping track of the acyclicity of the tournament. Observe also, that our parameter is now $\text{FVST}(T)$, which behaves in a different way than $\text{FAST}(T)$. For instance, our triangle detection data structures rely on the fact, that the number of back arcs in a tournament T is bounded in terms of $\text{FAST}(T)$. This, unfortunately, stops to be the case: the number of back arcs can be actually unbounded in terms of $\text{FVST}(T)$. In the following sections, we describe how our data structure deals with all these issues.

4.1 Vertex Removals and Restorations

We first deal with the problem of removing and restoring vertices. In Appendix C.2 in [28] we introduce a new data structure called $\text{DREM}[n, k]$, which essentially extends the data structure $\text{DTP}[n]$ by the possibility of removing and restoring up to k vertices. This data structure is covered in detail by Lemma 59 in Appendix C.2 in [28], below we only present the main ideas.

The data structure $\text{DREM}[n, k]$ maintains all the information about the tournament T that was maintained by $\text{DTP}[n]$, with the exception of the set **Back** of back arcs, whose maintenance now becomes prohibitively expensive. In addition to that, $\text{DREM}[n, k]$ maintains



■ **Figure 3** Implicit representation of tournament T^{-F} .

the set of removed vertices F and an implicit representation of T^{-F} . Here, T^{-F} stands for the tournament T with the set F of vertices removed, i.e, $T^{-F} = T[V(T) \setminus F]$. As mentioned before, we need to be able to quickly decrease/increase indegrees of many vertices, because removal/restoration of a vertex can pessimistically affect all other vertices. In order to do so, the $\mathbb{D}\text{REM}[n, k]$ data structure stores a set of token positions. One token corresponds to one removed vertex. Each time a vertex is removed, some token is placed at some position d . The vertices are again partitioned into buckets. A token at position d decreases by one the indegrees of vertices in all buckets whose indices are at least d . An illustration of the intended role of tokens can be found in Figure 3.

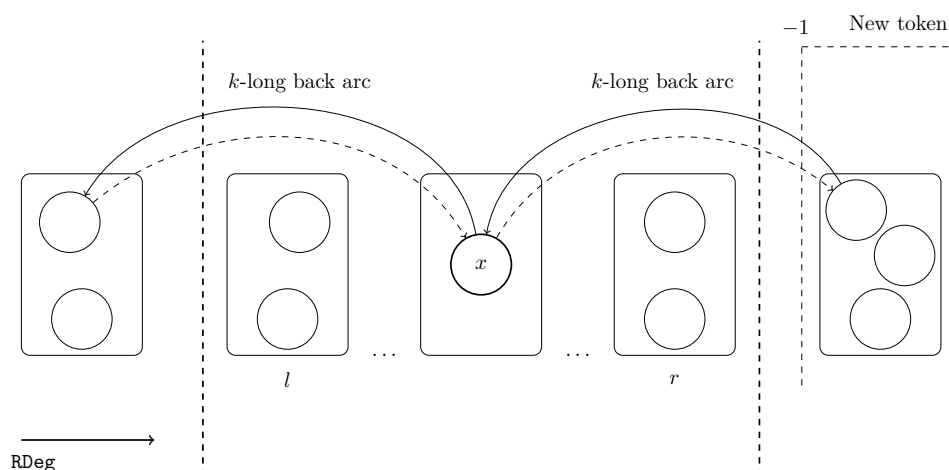
Ideally, to maintain T^{-F} , we would like to partition the vertices into buckets according to indegrees in T^{-F} . Due to the presence of tokens this is not feasible. Instead, we introduce a reduced indegree $\text{RDeg}[v]$ of a vertex v , which relates indegree $d_{T^{-F}}(v)$ of a vertex v in T^{-F} with token positions. To be more precise, let us denote by $\text{CTok}(d)$ the number of tokens at positions smaller or equal than d . The reduced indegree of a vertex v satisfies

► **Invariant 1.** $\text{RDeg}[v] - \text{CTok}(\text{RDeg}[v]) = d_{T^{-F}}(v)$.

This is an invariant of the data structure which guarantees, that the tokens reflect their intended role. Rather than according to indegrees in T^{-F} , the vertices of T^{-F} are then partitioned according to their reduced indegrees, and we refer to this partition as *reduced indegree buckets*. The reduced indegrees and the partition into reduced indegree buckets are maintained by $\mathbb{D}\text{REM}[n, k]$ in order to implicitly maintain tournament T^{-F} . This is illustrated in Figure 3. Similarly as before, also the set of empty reduced indegree buckets is maintained.

Apart from T , F , tokens, and the implicit representation of tournament T^{-F} , the $\mathbb{D}\text{REM}[n, k]$ data structure stores all k -long back arcs in T . For an arc uv we define its length with respect to tournament T as $l_T(uv) = |d_T(u) - d_T(v)|$. An arc is k -long if its length is at least k , otherwise the arc is k -short. In the $\mathbb{D}\text{REM}[n, k]$ data structure, every vertex stores a set $\text{LONG}(k, v)$ of k -long back arcs (with regard to T) adjacent to it.

The vertex removal procedure is illustrated in Figure 4. It is based on a method $\text{NewRd}(u)$, which given any vertex u , recomputes its reduced degree to restore Invariant 1. Given the set of removed vertices F , a list of token positions, and indegrees in T , one can compute $d_{T^{-F}}(u)$ and the value $\text{RDeg}[u]$ satisfying Invariant 1 in $O(|F|) = O(k)$ time. Due to space constraints, the details are given in the proof of Lemma 59 in Appendix C.2 in [28].



■ **Figure 4** Vertex removal/restoration procedure.

In order to remove (or restore) a vertex v , we need to fix the reduced degrees of all affected vertices in order to satisfy Invariant 1. To accomplish that, we first iterate over vertices u having their reduced indegrees inside a small interval $[l, r]$ around the reduced indegree of v , and we fix all these vertices using $\text{NewRd}(u)$. The size of the interval is bounded by $r - l \in O(k)$, so this takes time proportional to k^2 multiplied by maximum size of a reduced indegree bucket. Observe that $\text{RDeg}[v]$, $d_{T-F}(v)$ and $d_T(v)$ differ from each other by at most $O(k)$. Thus, the interval $[l, r]$ can be chosen in a way that all arcs between v and vertices with reduced degrees outside of $[l, r]$ are k -long arcs with regard to T (for an illustration see Figure 4). Thus, for all vertices w such that $\text{RDeg}[w] < l$, whose indegree changes as a result of v 's removal (or restoration), it holds that vw is a k -long back arc with regard to T . By iterating through $\text{LONG}(k, v)$ we can detect such vertices and fix their reduced indegrees. On the other hand, for all vertices w such that $\text{RDeg}[w] > r$, their indegree does decrease by one after v 's removal (or increase after v 's restoration), unless wv is a k -long back arc with regard to T . Thus, placing a token at position $r + 1$ fixes the invariant for all vertices with $\text{RDeg}[w] > r$, with the exception of the endpoints of k -long back arcs whose other endpoint is v . We can detect these vertices by iterating through $\text{LONG}(k, v)$ and fix their reduced indegrees. To sum up, the runtime of removal/restoration of a vertex v depends on $|\text{LONG}(k, v)|$ and on k^2 multiplied by the maximum size of a reduced indegree bucket. In Subsection 4.3 we show how to bound these parameters in terms of $\text{FVST}(T)$ to guarantee efficient running times in the promise model.

4.2 Detecting Triangles

Next, using the $\text{DREM}[n, k]$ data structure, we need to implement triangle detection in T^{-F} . To achieve that, we follow ideas from Section 2. Recall, that in order to find a triangle in a tournament T , we needed a method to find v_{\min} - a vertex of minimum indegree in $T^{-\text{pref}}$ and we needed a method for listing in-neighbours. We showed that this can be done in time proportional to $|\text{Empty}|$ and $|\text{Back}|$. In fact, we did not need to iterate through all back arcs, but just the ones incident to a vertex whose in-neighbours we seek. We now want to run these procedures on T^{-F} instead of T . We can access T^{-F} via the reduced indegree buckets stored by $\text{DREM}[n, k]$ data structure. In Appendix C.4 in [28] we carefully show, that the reduced degree buckets are functionally very close to the indegree buckets of T^{-F} , and we

can essentially use them instead to find v_{\min} and list in-neighbours with regard to T^{-F} . Still, we need reasonable bounds in terms of $\text{FVST}(T)$ on $|\text{Empty}^{-F}|$ and $|\text{Back}_v^{-F}|$, which are the number of empty indegree buckets in T^{-F} and the number of back arcs incident to a vertex v in T^{-F} . We show how to bound these in the next subsections.

4.3 Bounds

We first observe that the maximum size of an indegree bucket in a tournament T satisfies

► **Fact 11** (Lemma 55 in Appendix C.4 in [28]). $\max_{\text{Db}}(T) \leq 2\text{FVST}(T) + 1$.

To see why this holds, consider an indegree bucket $\text{Db}_T[d]$ of maximum size. Let $S = \text{Db}_T[d] \setminus F_V$, where F_V is the feedback vertex set of minimum size. After removal of F_V , each vertex in S lands in a separate indegree bucket. This implies that one of the vertices in S decreases its indegree by at least $|S| - 1$ after removing F_V . Each vertex removal decreases indegree of any other vertex by at most one. Thus, $|S| - 1 \leq |F_V|$ and $|\text{Db}_T[d]| \leq |S| + |F_V| \leq 2|F_V| + 1$.

Let now $\max_{\text{RDb}}(T, F)$ denote the maximum size of a reduced indegree bucket. We can relate $\max_{\text{RDb}}(T, F)$ to $\max_{\text{Db}}(T)$ as follows.

► **Fact 12** (Observation 61 in Appendix C.4 in [28]). $\max_{\text{RDb}}(T, F) \leq \max_{\text{Db}}(T)(|F| + 1)$

The reason why Fact 12 holds is as follows. As Figure 3 suggests, any reduced degree bucket is entirely contained in some indegree bucket with regard to T^{-F} . The vertices of one indegree bucket in T^{-F} are contained in at most $|F| + 1$ indegree buckets in T , as any vertex can decrease its indegree by at most $|F|$ after removing F .

Let $\text{LONG}(k, X)$ for $X \subseteq V(T)$ denote a set of k -long back arcs (with regard to T) between the vertices in X . Slightly more elaborate arguments of similar nature as above allow us to bound $|\text{Empty}^{-F}|$ as follows.

► **Fact 13** (Lemma 67 and Corollary 68 in Appendix C.4 in [28]).

$$|\text{Empty}^{-F}| \leq \text{FVST}(T^{-F}) \cdot (2(k + |F|) + |\text{LONG}(k, V(T^{-F}))|) + 2\text{FVST}(T^{-F}) + 5) + 4|F|$$

Given the above bound, our goal is now to bound $|\text{LONG}(k, V(T^{-F}))|$. As Section 4.1 and Section 4.2 suggest, we also need to bound $|\text{Back}_v^{-F}|$ and $|\text{LONG}(k, v)|$ for all $v \in V(T^{-F})$.

Due to Fact 12 and a close relation between reduced degree buckets and indegree buckets of T^{-F} , there is a bounded number of $(k + |F|)$ -short arcs in Back_v^{-F} . Since removing a vertex can decrease the indegree of any other vertex by at most one, the arcs that are $(k + |F|)$ -long in T^{-F} are k -long in T . Thus, bounding $|\text{Back}_v^{-F}|$ for $v \in T^{-F}$ boils down to bounding $|\text{LONG}(k, v)|$ for $v \in V(T^{-F})$.

To sum up, it suffices that we bound $|\text{LONG}(k, V(T^{-F}))|$ and $|\text{LONG}(k, v)|$ for all $v \in V(T^{-F})$. We deal with this in the subsequent section.

4.4 Kernelization Technique

In order to bound $|\text{LONG}(k, V(T^{-F}))|$ and $|\text{LONG}(k, v)|$ for $v \in V(T^{-F})$, we actually reduce the number of k -long back arcs by removing some vertices from T . To achieve this, we define a k -long graph G_{LONG} of the tournament T , which is an undirected graph, where vertices are connected via an edge in G_{LONG} if they are connected via a k -long back arc in the tournament T . We also define the k -heavy set $\text{Heavy}_k(T)$ of the tournament T as the set of vertices of degree higher than k in G_{LONG} . If $F_V \subseteq V(T)$ is a feedback vertex set in T of size at

most k , then F_V is necessarily a vertex cover in G_{LONG} : there is no other way to get rid of a k -long back arc from T than to remove its endpoint (see Lemma 64 in [28] for a more formal argument). Due to standard vertex cover kernelization arguments, $\text{Heavy}_k(T) \subseteq F_V$ for any feedback vertex set F_V of T with $|F_V| \leq k$. Moreover, when we remove $\text{Heavy}_k(T)$ from G_{LONG} , at most $k|F_V|$ edges remain in G_{LONG} . This simple but crucial observations are covered by Lemma 64 in [28]. So the idea is to keep the set $\text{Heavy}_k(T)$ removed from the tournament, in order to keep the number of k -long back arcs connecting the remaining vertices small. Observe also, that if $\text{Heavy}_k(T) \subseteq F$, where F is the set of removed vertices, any vertex that is left in T^{-F} has at most k adjacent k -long back arcs, what gives the desired bound on $|\text{LONG}(k, v)|$ for $v \in V(T^{-F})$. We want to emphasize, that the property of an arc being k -long is considered here with regard to tournament T and it does not depend on F , so we can maintain G_{LONG} in the form of adjacency lists efficiently.

To implement the above idea, in Appendix C.3 we introduce a wrapper data structure around the $\text{DREM}[n, k]$ called $\text{DREMP}[n, k]$. It allows only one kind of updates, arc reversals, and keeps the invariant that the k -heavy set of the maintained tournament is removed. This not only allows us to efficiently implement the methods for finding triangles, but also ensures fast running times of $\text{DREM}[n, k]$ operations in the promise model. The wrapper is defined in Lemma 66 in [28].

4.5 The Final Data Structure

In order to implement Algorithm 2, we use the $\text{DREMP}[n, k]$ data structure for $k = g(K)$, where K is the problem parameter. The $\text{DREMP}[n, k]$ data structure keeps $\text{Heavy}_k(T)$ removed from the tournament, i.e., $\text{Heavy}_k(T) \subseteq F$ at all times, where F is the set of currently removed vertices. This provides us with the bounds on $|\text{LONG}(k, v)|$ for $v \in T^{-F}$ and $|\text{LONG}(k, V(T^{-F}))|$, which we need to efficiently find triangles. When we branch on the vertices of the found triangle, we use the methods of the $\text{DREM}[n, k]$ data structure (internally maintained by $\text{DREMP}[n, k]$), in order to temporarily remove these vertices from the tournament. This approach leads to the following theorem.

► **Theorem 14** (Theorem 51 in Appendix C in [28]). *The dynamic K -FVST problem admits a data structure with initialization time $\mathcal{O}(n^2)$, worst-case update time $\mathcal{O}(g(K)^5)$ and worst-case query time $\mathcal{O}(3^K K^3 g(K))$ under the promise that there is a computable function g , such that tournament T always has a feedback vertex set of size at most $g(K)$.*

References

- 1 Josh Alman, Matthias Mnich, and Virginia Vassilevska Williams. Dynamic parameterized problems and algorithms. *ACM Trans. Algorithms*, 16(4), July 2020. doi:10.1145/3395037.
- 2 Noga Alon. Ranking tournaments. *SIAM Journal on Discrete Mathematics*, 20(1):137–142, 2006. doi:10.1137/050623905.
- 3 Noga Alon, Daniel Lokshtanov, and Saket Saurabh. Fast FAST. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikolettseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*, volume 5555 of *Lecture Notes in Computer Science*, pages 49–58. Springer, 2009. doi:10.1007/978-3-642-02927-1_6.
- 4 Jeffrey Banks. Sophisticated voting outcomes and agenda control. Working Papers 524, California Institute of Technology, Division of the Humanities and Social Sciences, 1984. URL: <https://EconPapers.repec.org/RePEc:clt:sswopa:524>.

- 5 Jiehua Chen, Wojciech Czerwiński, Yann Disser, Andreas Emil Feldmann, Danny Hermelin, Wojciech Nadara, Marcin Pilipczuk, Michał Pilipczuk, Manuel Sorge, Bartłomiej Wróblewski, and Anna Zych-Pawlewicz. Efficient fully dynamic elimination forests with applications to detecting long paths and cycles. In *Proceedings of the Thirty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '21*, pages 796–809, USA, 2021. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611976465.50.
- 6 Don Coppersmith, Lisa K. Fleischer, and Atri Rurda. Ordering by weighted number of wins gives a good ranking for weighted tournaments. *ACM Trans. Algorithms*, 6(3), July 2010. doi:10.1145/1798596.1798608.
- 7 Zdenek Dvořák, Martin Kupec, and Vojtech Tůma. A dynamic data structure for MSO properties in graphs with bounded tree-depth. In *22th Annual European Symposium on Algorithms, ESA 2014*, volume 8737 of *Lecture Notes in Computer Science*, pages 334–345. Springer, 2014. doi:10.1007/978-3-662-44777-2_28.
- 8 Zdenek Dvořák and Vojtech Tůma. A dynamic data structure for counting subgraphs in sparse graphs. In *13th International Symposium on Algorithms and Data Structures, WADS 2013*, volume 8037 of *Lecture Notes in Computer Science*, pages 304–315. Springer, 2013. doi:10.1007/978-3-642-40104-6_27.
- 9 Pál Erdős and John W. Moon. On sets of consistent arcs in a tournament. *Canadian Mathematical Bulletin*, 8:269–271, 1965. URL: <https://api.semanticscholar.org/CorpusID:19010097>.
- 10 Uriel Feige. Faster fast(feedback arc set in tournaments), 2009. arXiv:0911.5094.
- 11 W Fernandez de la Vega. On the maximum cardinality of a consistent set of arcs in a random tournament. *Journal of Combinatorial Theory, Series B*, 35(3):328–332, 1983. doi:10.1016/0095-8956(83)90060-6.
- 12 Fedor V. Fomin and Michał Pilipczuk. On width measures and topological problems on semi-complete digraphs. *Journal of Combinatorial Theory, Series B*, 138:78–165, 2019. doi:10.1016/j.jctb.2019.01.006.
- 13 Serge Gaspers and Matthias Mnich. Feedback vertex sets in tournaments. In Mark de Berg and Ulrich Meyer, editors, *Algorithms – ESA 2010*, pages 267–277, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. doi:10.1007/978-3-642-15775-2_23.
- 14 Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, July 2001. doi:10.1145/502090.502095.
- 15 Yoichi Iwata and Keigo Oka. Fast dynamic graph algorithms for parameterized problems. In *14th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2014*, volume 8503 of *Lecture Notes in Computer Science*, pages 241–252. Springer, 2014. doi:10.1007/978-3-319-08404-6_21.
- 16 Claire Kenyon-Mathieu and Warren Schudy. How to rank with few errors. In *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing, STOC '07*, pages 95–103, New York, NY, USA, 2007. Association for Computing Machinery. doi:10.1145/1250790.1250806.
- 17 Tuukka Korhonen, Konrad Majewski, Wojciech Nadara, Michał Pilipczuk, and Marek Sokołowski. Dynamic treewidth, 2023. arXiv:2304.01744.
- 18 R. Krithika, Abhishek Sahu, Saket Saurabh, and Meirav Zehavi. The parameterized complexity of packing arc-disjoint cycles in tournaments, 2018. arXiv:1802.07090.
- 19 Mithilesh Kumar and Daniel Lokshantov. Faster Exact and Parameterized Algorithm for Feedback Vertex Set in Tournaments. In Nicolas Ollinger and Heribert Vollmer, editors, *33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016)*, volume 47 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 49:1–49:13, Dagstuhl, Germany, 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2016.49.
- 20 Daniel Lokshantov, Pranabendu Misra, Joydeep Mukherjee, Fahad Panolan, Geevarghese Philip, and Saket Saurabh. 2-approximating feedback vertex set in tournaments. *ACM Trans. Algorithms*, 17(2), April 2021. doi:10.1145/3446969.

- 21 Konrad Majewski, Michał Pilipczuk, and Marek Sokołowski. Maintaining CMSO2 Properties on Dynamic Structures with Bounded Feedback Vertex Number. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science (STACS 2023)*, volume 254 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 46:1–46:13, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2023.46.
- 22 Matthias Mnich, Virginia Vassilevska Williams, and László A. Végh. A $7/3$ -Approximation for Feedback Vertex Sets in Tournaments. In Piotr Sankowski and Christos Zaroliagis, editors, *24th Annual European Symposium on Algorithms (ESA 2016)*, volume 57 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 67:1–67:14, Dagstuhl, Germany, 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2016.67.
- 23 Jędrzej Olkowski, Michał Pilipczuk, Mateusz Rychlicki, Karol Węgrzycki, and Anna Zych-Pawlewicz. Dynamic Data Structures for Parameterized String Problems. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science (STACS 2023)*, volume 254 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 50:1–50:22, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2023.50.
- 24 Venkatesh Raman and Saket Saurabh. Parameterized algorithms for feedback set problems and their duals in tournaments. *Theoretical Computer Science*, 351(3):446–458, 2006. Parameterized and Exact Computation. doi:10.1016/j.tcs.2005.10.010.
- 25 K.B. Reid and E.T. Parker. Disproof of a conjecture of Erdős and Moser on tournaments. *Journal of Combinatorial Theory*, 9(3):225–238, 1970. doi:10.1016/S0021-9800(70)80061-8.
- 26 E. Speckenmeyer. On feedback problems in digraphs. In *Proceedings of the Fifteenth International Workshop on Graph-Theoretic Concepts in Computer Science*, WG '89, pages 218–231, Berlin, Heidelberg, 1990. Springer-Verlag.
- 27 Anke van Zuylen, Rajneesh Hegde, Kamal Jain, and David P. Williamson. Deterministic pivoting algorithms for constrained ranking and clustering problems. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 405–414, USA, 2007. Society for Industrial and Applied Mathematics. URL: <http://dl.acm.org/citation.cfm?id=1283383.1283426>.
- 28 Anna Zych-Pawlewicz and Marek Żochowski. Dynamic parameterized feedback problems in tournaments, 2024. doi:10.48550/arXiv.2404.12907.

Parameterized Shortest Path Reconfiguration

Nicolas Bousquet ✉ 

Univ. Lyon, LIRIS, CNRS, Université Claude Bernard Lyon 1, Villeurbanne, France

Kshitij Gajjar ✉

Centre for Security, Theory & Algorithmic Research (CSTAR),
International Institute of Information Technology, Hyderabad (IIIT-H), India

Abhiruk Lahiri ✉ 

Department of Computer Science, Heinrich Heine University, Düsseldorf, Germany

Amer E. Mouawad ✉

Department of Computer Science, American University of Beirut, Lebanon

Abstract

An st -shortest path, or st -path for short, in a graph G is a shortest (induced) path from s to t in G . Two st -paths are said to be adjacent if they differ on exactly one vertex. A reconfiguration sequence between two st -paths P and Q is a sequence of adjacent st -paths starting from P and ending at Q . Deciding whether there exists a reconfiguration sequence between two given st -paths is known to be PSPACE-complete, even on restricted classes of graphs such as graphs of bounded bandwidth (hence pathwidth). On the positive side, and rather surprisingly, the problem is polynomial-time solvable on planar graphs. In this paper, we study the parameterized complexity of the SHORTEST PATH RECONFIGURATION (SPR) problem. We show that SPR is $W[1]$ -hard parameterized by $k + \ell$, even when restricted to graphs of bounded (constant) degeneracy; here k denotes the number of edges on an st -path, and ℓ denotes the length of a reconfiguration sequence from P to Q . We complement our hardness result by establishing the fixed-parameter tractability of SPR parameterized by ℓ and restricted to nowhere-dense classes of graphs. Additionally, we establish fixed-parameter tractability of SPR when parameterized by the treedepth, by the cluster-deletion number, or by the modular-width of the input graph.

2012 ACM Subject Classification Theory of computation \rightarrow Parameterized complexity and exact algorithms

Keywords and phrases combinatorial reconfiguration, shortest path reconfiguration, parameterized complexity, structural parameters, treedepth, cluster deletion number, modular width

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.23

Related Version *Full Version*: <https://doi.org/10.48550/arXiv.2406.12717>

Funding *Abhiruk Lahiri*: Partly supported by SFF grant “Theoretical AI” of HHU Düsseldorf and the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 456558332.

1 Introduction

Many algorithmic questions can be posed as follows: given the description of a system state and the description of a state we would “prefer” the system to be in, is it possible to transform the system from its current state into a more desired one without “breaking” the system in the process? And if yes, how many steps are needed? Such problems naturally arise in the fields of mathematical puzzles, operational research, computational geometry [15], bioinformatics, and quantum computing [10]. These questions received a substantial amount of attention under the so-called *combinatorial reconfiguration framework* in the last decade. We refer the reader to the surveys by van den Heuvel [18], Nishimura [16] and Bousquet *et al.* [6] for more background on combinatorial reconfiguration.



© Nicolas Bousquet, Kshitij Gajjar, Abhiruk Lahiri, and Amer E. Mouawad;
licensed under Creative Commons License CC-BY 4.0

19th International Symposium on Parameterized and Exact Computation (IPEC 2024).

Editors: Édouard Bonnet and Paweł Rzażewski; Article No. 23; pp. 23:1–23:14

Leibniz International Proceedings in Informatics



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

Shortest path reconfiguration. In this work, we focus on the reconfiguration of st -shortest paths (or st -paths for short) in undirected, unweighted, simple graphs. It is well-known that one can easily find an st -path in a graph in polynomial time. In order to define the reconfiguration variant of the problem, we first require a notion of adjacency between st -paths.

As is common in the combinatorial reconfiguration framework, we focus on two models; the token-jumping model (TJ) and the token-sliding model (TS). We say that two st -paths are *TJ-adjacent* if they differ on exactly one vertex, i.e., all the vertices are the same except at a unique position p . We say that two st -paths P and Q are *TS-adjacent* if they are TJ-adjacent and the p^{th} vertex of P and the p^{th} vertex of Q are adjacent. A *reconfiguration sequence from P to Q* (if it exists) is a sequence of adjacent shortest paths starting at P and ending at Q . In the SHORTEST PATH RECONFIGURATION (SPR) problem, we are given a graph G , two vertices s and t , two st -paths P and Q of length k each, and the goal is to decide whether a reconfiguration sequence from P to Q exists. In the SHORTEST SHORTEST PATH RECONFIGURATION (SSPR) problem, we are additionally given an integer ℓ which is an upper bound on the length of the desired reconfiguration sequence. Reconfiguration of shortest paths has many applications, e.g., in network design and operational research (we refer the interested reader to [9] for a detailed discussion around these applications).

Many reconfiguration problems, SPR and SSPR included, naturally lie in the class PSPACE. Since there are no simple polynomial-time checkable certificates (as reconfiguration sequences are possibly of exponential length), they are generally not in NP. A decade ago, Bonsma [3] proved that SPR (under token jumping) is PSPACE-complete. In fact, the problem remains PSPACE-complete even when restricted to bipartite graphs [3], line graphs [9], and graphs of bounded bandwidth/pathwidth/treewidth [19]. Several groups studied the complexity of the problem in other restricted graph classes such as grid graphs [1], claw-free graphs, chordal graphs [3], and circle graphs [9]. The most notable result has been obtained by Bonsma who showed that SHORTEST PATH RECONFIGURATION can be decided in polynomial time for planar graphs [4]. This result is rather surprising in the reconfiguration setting since most reconfiguration problems are known to be PSPACE-complete on planar graphs, see e.g. [13, 14, 5].

Our results. Our focus is on the parameterized complexity of shortest path reconfiguration problems; which, to the best of our knowledge, has not been studied so far. Other reconfiguration problems have been widely studied from a parameterized perspective in the last decade, see, e.g., [6] for a survey. A problem is *fixed-parameter tractable*, FPT for short, on a class \mathcal{C} of graphs with respect to a parameter κ , if there is an algorithm deciding whether a given input instance with graph $G \in \mathcal{C}$ admits a solution in time $f(\kappa) \cdot |V(G)|^c$, for a computable function f and constant c .

A *kernelization algorithm* is a polynomial-time algorithm that reduces an input instance to an equivalent instance of size bounded in the parameter only (independent of the input size), known as a *kernel*; we will say that two instances are *equivalent* if they are both yes-instances or both no-instances. Every fixed-parameter tractable problem admits a kernel, however, possibly of exponential or worse size. For efficient algorithms, it is therefore most desirable to obtain polynomial, or even linear, kernels. The W-hierarchy is a collection of parameterized complexity classes $\text{FPT} \subseteq \text{W}[1] \subseteq \text{W}[2] \subseteq \dots \subseteq \text{W}[t]$, for $t \in \mathbb{N}$. The conjecture $\text{FPT} \subsetneq \text{W}[1]$ can be seen as the analogue of the conjecture that $\text{P} \subsetneq \text{NP}$. Before stating our results precisely, let us formally define the problems we are interested in (we intentionally omit the type of move, i.e., slide or jump, from the definitions, as it will be clear from context in what follows):

SHORTEST PATH RECONFIGURATION (SPR)

Input: A graph G , two vertices s, t , two st -shortest paths P, Q .

Question: Is there a reconfiguration sequence from P to Q ?

SHORTEST SHORTEST PATH RECONFIGURATION (SSPR)

Input: A graph G , two vertices s, t , two st -shortest paths P, Q , an integer ℓ .

Question: Is there a reconfiguration sequence from P to Q of length at most ℓ ?

In parameterized complexity, one is usually interested in two types of parameters: parameters related to the size of the solution or parameters related to the structure of the input graph. For shortest path reconfiguration, there are two parameters related to the size of the solution which are the length ℓ of a reconfiguration sequence, and the length k of the shortest st -paths (number of edges on the shortest st -paths) in G . Our first results will focus on these parameters. We will then discuss some parameters related to the graph structure such as treedepth and modular width. Our first result is a hardness result. We prove that the following holds (in both the token jumping and the token sliding models):

► **Theorem 1.** *SPR is $W[1]$ -hard parameterized by k , and SSPR is $W[1]$ -hard parameterized by $k + \ell$, in both the token jumping and the token sliding models.*

The idea of the proof of Theorem 1 is a reduction from the MULTICOLORED CLIQUE problem. Let $(V_i)_{i \leq k}$ be the vertices of an instance of the MULTICOLORED CLIQUE problem. Intuitively (the real proof being more technical), we will construct a graph where the length of the st -paths will be in $\mathcal{O}(k^2)$, each integer representing a vertex of the set V_i . The goal would be to transform a path P into a path Q , forcing us to select a vertex in each set. For every pair i, j , there exists an integer r such that the r^{th} vertex corresponds to a vertex in V_i and the $(r + 1)^{\text{th}}$ vertex corresponds to a vertex in V_j . The key argument of the proof consists in finding a mechanism to ensure that the vertex selected in each copy of V_i is the same, which permits us to conclude that the subset of selected vertices is a multicolored clique of the desired size. One can then naturally wonder if this hardness result can be pushed further. The answer is yes, and in fact, we prove (in the full version of the paper) that the problems are hard even restricted to a very simple class of graphs:

► **Theorem 2.** *SPR is $W[1]$ -hard parameterized by k , and SSPR is $W[1]$ -hard parameterized by $k + \ell$, even when the inputs are restricted to graphs of constant degeneracy and in both the token jumping and the token sliding models.*

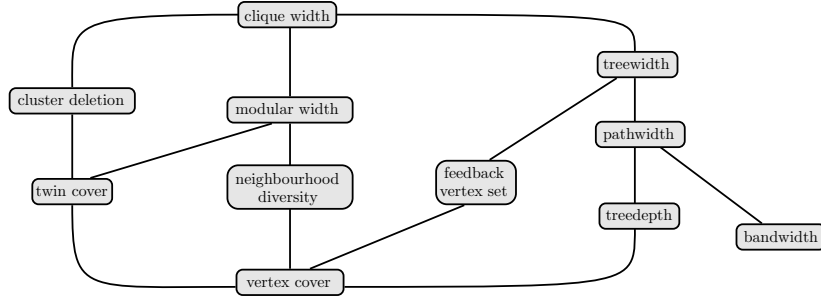
In order to prove that statement we adapt the proof of Theorem 1 to appropriately reduce the degeneracy of the graph. We then complement these negative results with the following positive ones.

► **Theorem 3.** *SSPR is FPT parameterized by ℓ on nowhere-dense classes of graphs (in both the token jumping and the token sliding models).*

The idea of the proof of Theorem 3 consists in proving that if k is too large compared to ℓ then there are many positions along the shortest paths that are already occupied by tokens that never have to move. Using this fact, we then contract parts of the paths in order to get st -paths of length $\mathcal{O}(f(\ell))$, for some computable function f . Now, since k is bounded by some function of ℓ , one can prove that the existence of a reconfiguration sequence of length ℓ can be verified via model checking a first-order formula ϕ whose size depends only on ℓ . Combining this observation with the black-box result of [11] that ensures that the model checking problem can be decided in time $\mathcal{O}(f(|\phi|) \cdot |V(G)|)$ on nowhere-dense graphs, we get the desired result. We proceed (in the full version of the paper) by considering some of the most commonly studied structural graph parameters. In particular, we prove the following:

► **Theorem 4.** *SPR and SSPR (in both the token jumping and the token sliding model) are FPT when parameterized by either the treedepth, the cluster deletion number, or the modular width of the input graph.*

To motivate the study of these parameters, we refer the reader to Figure 1. Recall that SPR is PSPACE-complete even when restricted to graphs of bounded bandwidth, pathwidth, treewidth, and cliquewidth [19]. This implies para-PSPACE-hardness on the aforementioned classes. Hence, our Theorem 4 almost completes the picture for structural parameterizations of the problems, leaving open the case of feedback vertex set number.



■ **Figure 1** The graph parameters studied in this paper. A connection between two parameters indicates the existence of a function in the one above that lower-bounds the one below.

Further discussions and open problems. As we show in the full version of the paper, it turns out that when solving the SPR problem parameterized by the feedback vertex set number of the graph, one can assume that k , the length of st -paths, is bounded linearly in the parameter. Hence, the following remains an interesting open question:

► **Problem 1.** *Is SPR fixed-parameter tractable when parameterized by feedback vertex set number?*

When the feedback vertex set number is bounded, the graph can be seen as a disjoint union of trees plus a bounded number of additional vertices. One can easily remark that if vertices of the feedback vertex set are far apart in the st -paths then the structure is very rigid and very few tokens can move in the graph. However, when vertices of the feedback vertex set are close to one another (along the st -paths), there might exist some arbitrarily long paths between two layers in the layered partition of the graph. Here, the layered partition refers to the partitioning of the vertex set based on distance either from s or from t . Tokens along these (layer) paths that do not belong to the feedback vertex set are not restricted and can traverse their corresponding layer path in both directions an unbounded number of times. In particular, it implies that, if there exists a reconfiguration sequence, that sequence might be arbitrarily long. So in order to design a reconfiguration sequence (from a kernelization perspective at least, which is known to be equivalent to fixed-parameter tractability), we have to find a way to reduce these long structures into structures of bounded length. We were not able to solve this very special case of the problem.

As far as we know, it also remains an open question whether SPR is in P or is NP-complete on graphs of constant feedback vertex set number. Note that an XP algorithm follows immediately from the fact that (after appropriately discarding parts of the input) the number of st -paths is roughly $|V(G)|^f$, where f denotes the feedback vertex set number. Regardless, in case of a positive answer to Problem 1, the next natural question is the following:

► **Problem 2.** *Is SPR fixed-parameter tractable when parameterized by k on graphs of bounded pathwidth? What about treewidth? How about parameterization by k plus the treewidth?*

It is an easy exercise to remark that SPR is PSPACE-complete on graphs of bounded bandwidth, pathwidth, and treewidth using a simple reduction from *H-WORD RECONFIGURATION* [19]. When the treewidth is 1, there exists a unique minimum st -path and the problem is simple. Trees and forests are graphs which are 1-degenerate and every 1-degenerate graph is a forest, however, the complexity of SPR and SSPR remains open for 2-degenerate graphs.

► **Problem 3.** *What is the complexity of SPR and SSPR on 2-degenerate graphs?*

Related work. Reconfiguration of paths and other subgraphs has been considered before [7, 12, 8]. For some of these past works, i.e., [7, 12], the paths have fixed length and a “move” consists of removing a vertex at one end and adding a vertex at the other end, as in certain “snake-like” games. In contrast, for our work, the two endpoints of the paths are fixed and the paths are required to be unweighted shortest paths between those endpoints.

Demaine *et al.* proved in [7] that the problem of reconfiguring (arbitrary) paths via snake-like moves is PSPACE-complete in general, and polynomial-time solvable for some restricted graph classes. When not restricted to shortest paths, the problem is quite different, since the extremities of the paths are not fixed and the goal is not necessarily to reconfigure shortest paths. In fact, it is proved in [7] that fixed-length path reconfiguration (under the snake-like moves described above) is fixed-parameter tractable parameterized by the path length or by the circuit rank, XP parameterized by the feedback vertex set number, and PSPACE-complete even for graphs of bounded bandwidth [19]. Gupta *et al.* [12] also show fixed-parameter tractability parameterized by path length for a different type of snake-like moves, i.e., paths are considered directed paths and are required to move forwards only.

Reconfiguration problems on graphs of bounded feedback vertex set number and on graphs of bounded treewidth have already received a considerable amount of attention, and they are usually not easy to place in FPT (unlike their optimization counterparts, where a simple branching strategy or dynamic programming algorithm is usually enough to get an FPT algorithm). For instance, INDEPENDENT SET RECONFIGURATION (in the token sliding model) on graphs of bounded feedback vertex set number is FPT; this fact follows easily from the multi-component reduction in [2]. However, the question is still open for the reconfiguration of dominating sets, for instance. The case of bounded treewidth graphs is open for both INDEPENDENT SET RECONFIGURATION and DOMINATING SET RECONFIGURATION (in the sliding model) [6].

2 Hardness results

We start with the case of SPR parameterized by k on general graphs. The same reduction will imply the hardness of SSPR parameterized by $k + \ell$. We describe in the full version of the paper how to modify the construction to obtain a graph of constant degeneracy¹.

Our reduction is from the REGULAR MULTICOLORED CLIQUE (RMC) problem, which is known to be NP-complete and W[1]-hard when parameterized by solution size κ [17]. The problem is defined as follows. We are given a κ -partite graph $G = (V, E)$ such that V is partitioned into κ independent sets $V = V_1 \cup V_2 \cup \dots \cup V_\kappa$ and each partition has size exactly n , i.e., $|V| = \kappa n$. We denote the vertices of V_i by $v_1^i, v_2^i, \dots, v_n^i$. Moreover, every vertex

¹ Proofs of statements marked with a star are omitted due to space constraints.

$v_j^i \in V_i$ has exactly r neighbors in every set $V_{i'}$, $i \neq i'$. In other words, every vertex in G has degree exactly $r(\kappa - 1)$. Given an instance (G, κ) of RMC, the goal is to decide if G contains a clique of size κ , which we call a multicolored clique since it must contain exactly one vertex from each V_i , $i \in [\kappa]$. We reduce (G, κ) to an instance (G', s, t, P, Q) of SPR, where P and Q are st -paths in G' of length $k = \mathcal{O}(\kappa^2)$.

Properly colored st -paths. Before discussing G' , we start by describing a key gadget of our construction which is a graph called H . The graph H consists of $\alpha = 6\kappa^2$ sets of vertices $H_1, H_2, \dots, H_\alpha$ such that $|H_i| = n$ for each $i \in [\alpha]$. We group every three consecutive sets into $\beta = 2\kappa^2$ groups $R_1 = \{H_1, H_2, H_3\}$, $R_2 = \{H_4, H_5, H_6\}$, $R_3 = \{H_7, H_8, H_9\}$, \dots , and $R_\beta = \{H_{\alpha-2}, H_{\alpha-1}, H_\alpha\}$. We call H_i the i th layer of H and R_i the i th group of H ; it will become clear later that a shortest path will select a vertex from each H_i . We also define a mapping $\mu : [\beta] \rightarrow [\kappa]$ such that each R_i is mapped to some V_j , for $i \in [\beta]$ and $j \in [\kappa]$. In other words, each $R_i = \{H_a, H_b, H_c\}$ will correspond to taking three copies of some V_j . We sometimes abuse notation and write $\mu(R_i) = V_j$ to denote the image of a set. We also overload notation and write $\mu(H_p) = V_j$ whenever $H_p \in R_i$ and $\mu(R_i) = V_j$.

Furthermore, we construct μ in such a way that, for every pair (j, j') , $j \neq j'$ and $j, j' \in [\kappa]$, there exists at least one integer $i < \beta$ such that $\mu(i) = j$, $\mu(i + 1) = j'$. In other words, for every two sets V_j and $V_{j'}$, there must exist two consecutive groups R_i and R_{i+1} such that R_i is mapped to V_j and R_{i+1} is mapped to $V_{j'}$. One can easily check that it is indeed possible to construct such a function μ when $\beta = 2\kappa^2$. We define μ as follows:

$$\text{For each } i \in [\beta], R_i \text{ is mapped to } V_{\mu(i)}, \text{ where } \mu(i) = \begin{cases} 1 + \lfloor (i-1)/2\kappa \rfloor & i \text{ is odd;} \\ 1 + ((i-2) \bmod 2\kappa)/2 & i \text{ is even.} \end{cases}$$

► **Observation 5.** For each $(j, j') \in [\kappa] \times [\kappa]$ such that $j \neq j'$, there exists an $i \in [\beta - 1]$ such that $\mu(i) = j$ and $\mu(i + 1) = j'$.

We also define a mapping $\pi_i : R_i \rightarrow V_{\mu(i)}$ (and $\pi_i : H_i \rightarrow V_{\mu(i)}$) that maps every vertex of R_i (H_i) to its corresponding vertex in $V_{\mu(i)}$. We drop the subscript i when clear from context. We note that each vertex of $V_{\mu(i)}$ appears three times in R_i (once in each layer) and all three vertices map to the same vertex of $V_{\mu(i)}$. Let us now describe the edge set of H . For every $i \in [\beta]$, we add a matching between vertices of H_j and H_{j+1} and a matching between vertices of H_{j+1} and H_{j+2} whenever there exists a group R_i such that $R_i = \{H_j, H_{j+1}, H_{j+2}\}$. For every two consecutive groups $R_i = \{H_j, H_{j+1}, H_{j+2}\}$ and $R_{i+1} = \{H_{j+3}, H_{j+4}, H_{j+5}\}$, we add in H the edges of G between H_{j+2} and H_{j+3} . That is, we add between consecutive sets corresponding to different sets of G the edges corresponding to the edges between those two sets in G . More formally, let $a \in H_{j+2}$, $b \in H_{j+3}$, $\pi(a) \in V_{\mu(i)}$, and $\pi(b) \in V_{\mu(i+1)}$. Then, there is an edge between vertices a and b in H if and only if there is an edge between vertices $\pi(a)$ and $\pi(b)$ in G .

Assume that we create a new graph H' consisting of H plus two additional vertices s and t , where s is connected to all the vertices of H_1 and t is connected to all the vertices of H_α . Note that any st -path in H' must contain exactly one vertex from every layer. We say that an st -path P is *properly colored* whenever for any $a \in H_i$ and $b \in H_j$ (on the path) such that $\mu(i) = \mu(j)$, we have $\pi(a) = \pi(b)$. In other words, whenever two layers of H (containing vertices of P) map to the same set of V we must select the same vertices in both. We note that any st -path P in H' can intersect with a group R_i in one of n ways, i.e., the vertices of P in R_i all map to the same vertex of $V_{\mu(i)}$.

► **Observation 6** (\star). H' contains a properly colored st -path P (consisting of $6\kappa^2 + 2$ vertices) if and only if G contains a multicolored clique of size κ .

Outline of the reduction. Assume that we add to the graph H' two new (internally) vertex-disjoint st -paths P and Q each containing exactly $\alpha + 2$ vertices (s and t and one vertex per layer of H). We add all the edges between the i -th vertex of P and the vertices in layers i , $i - 1$, and $i + 1$ of H (with the assumption that $H_0 = \{s\}$ and $H_{\alpha+1} = \{t\}$). Similarly, we add all the edges between the i -th vertex of Q and the vertices in layers i , $i - 1$, and $i + 1$ of H . We denote the resulting graph by $H' + P + Q$.

Consider the instance $(H' + P + Q, s, t, P, Q)$ of SPR. If there exists a multicolored clique in G then there exists a properly colored st -path in H' by Observation 6. By the definition of the edge set, one can easily see that we can transform P into Q by first moving the vertices of P onto a properly colored st -path in H' and then moving all the vertices to Q one by one. Unfortunately, the converse is not necessarily true since we might not be consistent in the selection of vertices in H' , i.e., we might select vertices $a \in H_i$ and $b \in H_j$ in the path such that $\mu(i) = \mu(j)$ and $\pi(a) \neq \pi(b)$ (H_i and H_j belong to different groups).

By considerably complicating the gadgetry, we will prove that we can handle this issue. To do so, we create a new gadget that will force us to select the same vertex for a fixed value of the image of μ . We replicate our gadget to enforce the consistency of all the images of μ . In addition to enforcing consistent selection of vertices, our construction further guarantees that choices cannot be undone.

Another issue in the simplistic construction of H' described above is that we implicitly assume that we move from P to a path fully contained in H before going to Q . But nothing prevents an st -path from containing some vertices of P , then some vertices from H , then some vertices from Q , then more vertices from H , and so on. To avoid this phenomenon, we shall add what we call buffer space. We formalize all these ideas next.

Buffers and collapses. Most of the time, we will consider matchings and edges between sets of size n . Given two sets of size n (with an implicit ordering), we define the natural matching as the matching that matches the vertices in increasing index order (in the natural way). We will sometimes consider edges between a set A of size n and a set B of size larger than n with a canonical mapping function to $\{1, \dots, n\}$. By abuse of notation, we still denote by the natural matching the set of edges (that is not a matching anymore) that links the i -th vertex of A and all the vertices that map to i in B .

We denote by I^n (J^n) the independent set on n vertices². We drop the superscript n when clear from context. We let \mathcal{I}^q (\mathcal{J}^q) denote the graph obtained by taking q copies of I^n (J^n) where consecutive copies of I^n (J^n) are linked with the natural matching. Note that \mathcal{I}^q (\mathcal{J}^q) consists of exactly n paths on q vertices. We use I_i (resp. J_i) to denote the i th copy of I^n (resp. J^n) in \mathcal{I}^q (resp. \mathcal{J}^q).

Let $R = R_1, R_2, \dots, R_\gamma$ be a graph where edges are between consecutive sets and there is a canonical mapping from R_1 and R_γ to $\{1, \dots, n\}$ (in our proof, R will be H or a graph close to H). We write $\Gamma(p, H, q) = \mathcal{I}^p \oplus H \oplus \mathcal{J}^q$ (or Γ when p, q, H are clear from context) to denote the graph obtained by taking a copy of \mathcal{I}^p , a copy of \mathcal{J}^q , a copy of H , and then adding the natural matching between the vertices of \mathcal{I}^p and H_1 as well as a matching between the vertices of H_α and J_1 . If we denote by I_i the sets of \mathcal{I}^p and J_i the sets of \mathcal{J}^q , for $i \in [p + \alpha + q]$, we call L_i the i -th layer of $\Gamma(p, H, q)$, where $L_i = I_i$ when $i \leq p$, $L_i = H_{i-q}$ when $p < i \leq p + \alpha$, and $L_i = J_{i-(q+\alpha)}$ when $i > p + \alpha$.

² These two notations that denote the same graph will permit to simplify the description of the constructions in the rest of the paper.

Given the graph H (recall that H consists of $\alpha = 6\kappa^2$ sets of vertices H_1, \dots, H_α) and a vertex $h_j^i \in H_i$, we let $H(h_j^i)$ denote the graph obtained from H by deleting all but one vertex from each set $H_{i'}$, where $\mu(i') = \mu(i)$; we delete all vertices except for $h_j^{i'} \in H_{i'}$ (deleting vertices implies the deletion of edges incident on those vertices). That is, we restrict all the layers of H corresponding to $V_{\mu(i)}$ to a single vertex (the same vertex); we have $\pi(h_j^i) = \pi(h_j^{i'})$ for all i, i' with $\mu(i) = \mu(i')$. We say that $H(h_j^i)$ is a *collapse of H on h_j^i* , or, equivalently, collapsing H on h_j^i results in $H(h_j^i)$.

Now, for every $i \leq \kappa, j \leq n$, we define $\Gamma_{i,j}(p, q)$ as $\Gamma(p, H(h_j^i), q)$. Finally, we let $\Gamma^i(p, q)$ denote the union of the n graphs $\Gamma_{i,j}(p, q)$. We write $\Gamma_{i,j} = \Gamma_{i,j}(p, q)$ whenever p and q are clear from context. Note that all the $\Gamma_{i,j}$ being disjoint, if we have a path fully included in one of the $\Gamma_{i,j}(p, q)$ at some point, then all the selected vertices in sets mapping to i by μ are the same. That is, $\Gamma^i(p, q)$ will allow us to verify that for any $H_j, H_{j'}$ in the selection gadget such that $\mu(j) = \mu(j') = i$ we always pick vertices $a \in H_j, b \in H_{j'}$ such that $\pi(a) = \pi(b)$.

Construction. We are now ready to describe the construction of the instance (G', s, t, P, Q) of SPR. We consider the token jumping model (changes required for sliding can be found in the full version of the paper). We start from an empty graph G' and add two new vertices s and t . We let $q = 2\kappa^2$ and $\delta = 2q + \alpha = 10\kappa^2$. We add two internally vertex-disjoint st -paths P and Q consisting of δ internal vertices each.

The next step consists of adding $\Gamma_\star = \Gamma(q, H, q)$ to G' and connecting s to every vertex in I_1 and t to every vertex in J_q . Moreover, we let the i th internal vertex of $P, i \geq 2$, be adjacent to every vertex in layer $i - 1$ of Γ_\star . We call Γ_\star the *selection gadget*. The rest of the gadgets will be verification and boundary gadgets that allow us to guarantee that properties similar to those in Observation 6 will hold.

We then create a graph $\Gamma^1(q - 1, q + 1)$ denoted by Γ^1 which will be the *verification gadget for $i = 1$* . We deal with the graphs of Γ^1 first (and slightly differently than the rest) as they require special attention given that they exist at the “boundary” of our construction. Notice that, in G' , all the graphs in Γ^1 are “shifted one position to the left with respect to Γ_\star ” (in the sense that the number of independent sets at the left has reduced by one), see Figure 2 for an illustration. In particular, the graph H of each $\Gamma_{1,j}, j \in [n]$, starts (or appears) one layer before the graph H in Γ_\star . We now describe the edges between Γ_\star and any $\Gamma_{1,j}$ (in Γ^1). Let L denote some layer of $\Gamma_{1,j}$ (ignoring the last layer) and let L' be the layer after L in Γ_\star . If L and L' correspond to independent sets (not sets of H) they are connected by the natural matching. Otherwise, we have two cases:

- If layer L of $\Gamma_{1,j}$ corresponds to a set H_p with $\mu(p) = 1$ then we deleted all vertices of L except for h_j^p (collapse). We connect h_j^p to its image in L' , which must exist since layer L' of Γ_\star corresponds to a set $H_{p'}$ with $\mu(p) = \mu(p') = 1$.
- Otherwise, we have the same number of vertices in L and L' and we add a matching between the pairs of vertices having the same image in G .

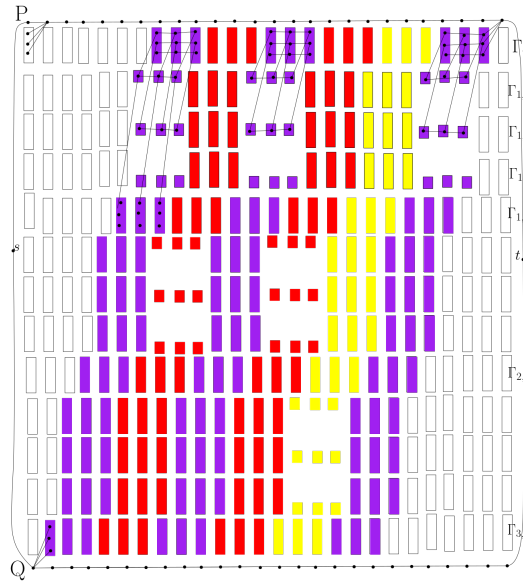
We now add a boundary gadget that will separate all the verification gadgets and allow us to simplify some of the arguments. Picturing the graph being constructed from top to bottom with P and Q encircling all of the graph, we assume that $\Gamma_{i,j}$ is drawn before $\Gamma_{i,j+1}$. Similarly, we only insert $\Gamma_{i+1,j}$ after inserting all graphs of Γ^i (see again Figure 2 for an illustration). After $\Gamma_{1,n}$ is inserted, we insert another graph (connecting s and t) that we denote by $\Gamma_{1,\star} = \Gamma(q - 2, H, q + 2)$ which is called the *boundary gadget of Γ^1* . Note that $\Gamma_{1,\star}$ is again shifted one position to the left compared to all the graphs in Γ^1 . We add edges between layers of $\Gamma_{1,\star}$ and layers of $\Gamma_{1,j}$, for each $j \in [n]$. Like before, we let L denote some

layer of $\Gamma_{1,\star}$ (ignoring the last layer) and let L' be the layer after L in $\Gamma_{1,j}$. If L and L' correspond to independent sets (not sets of an H) then we connect them via a matching in the natural way. Otherwise, we have again two cases:

- $|L| = n, |L'| = 1$, and we connect by an edge the unique vertex of L' to its image in L ; or
- $|L| = |L'| = n$ (by construction) and we connect the two layers by a matching.

We can now complete the construction as follows. For $i \in [\kappa - 1]$, after $\Gamma_{i,\star}$ is inserted we proceed just like before by assuming that $\Gamma_{i,\star}$ now takes the role of the selection gadget Γ_\star . Formally, for $i \in [\kappa - 1]$ and $j \in [n]$ (processing in increasing order), we create a graph $\Gamma_{i+1,j}$, where $\Gamma_{i+1,j} = \Gamma(q - (2i + 1), H(h_j^{i+1}), q + (2i + 1))$. We connect s to all the first-layer vertices and t to all the last-layer vertices in the obvious way. Let Γ^{i+1} denote the collection of the n graphs of the form $\Gamma_{i+1,j}$. We add edges between $\Gamma_{i,\star}$ and graphs in Γ^{i+1} just like before. Similarly, we then add a new graph $\Gamma_{i+1,\star}$ and proceed as described until we reach $\Gamma_{\kappa,\star}$. We connect all the vertices of a layer of $\Gamma_{\kappa,\star}$ to the vertex of Q on the preceding layer (see Figure 2). This completes the construction of the SPR instance (G', s, t, P, Q) ³. Note that $|V(P)| = |V(Q)| = 10\kappa^2 + 2$.

Safeness of the reduction. Before we dive into the technical details of the proof, let us give some high-level intuition. Simply put, the purpose of every set of graphs $\Gamma^i, i \in [\kappa]$, is to verify that all the sets/layers of Γ_\star mapping to the same V_i use the same vertex of V_i . The trickier part of the proof is in showing that tokens are “well-behaved”.



■ **Figure 2** An example of our reduction in the case of token jumping.

Let us start by proving the easier direction. We assume, without loss of generality, that all of our gadgets H start with a copy of V_1 and end with a copy of V_κ . Moreover no two consecutive groups of any H map to the same V_i .

³ We note that most of the buffer space “to the right” of the construction is not needed but was added to favor a symmetric construction.

23:10 Parameterized Shortest Path Reconfiguration

► **Lemma 7.** *If (G, κ) is a yes-instance of (REGULAR) MULTICOLORED CLIQUE then there exists a reconfiguration sequence from P to Q whose length is $20(\kappa^3 + \kappa^2)$.*

Proof. Let $\{v_{j_1}^1, v_{j_2}^2, \dots, v_{j_i}^i, \dots, v_{j_\kappa}^\kappa\}$ denote the vertices of a multicolored clique in G . Let us exhibit a reconfiguration sequence from P to Q . To do so, let us first give a reconfiguration sequence from P to a path that contains vertices in Γ_\star as follows: We move one by one the tokens of P to Γ_\star by increasing distance to s (in ascending order). For every layer $i \leq q$, we jump (in order) the token at layer $i \geq 1$ in P to vertex v_{j_1} in the i th layer of $\Gamma_\star = \Gamma(q, H, q)$ as long as $i \leq q + 1$ (as H_1 maps to V_1 by assumption). In other words, we map all the vertices at the beginning of the path to the copy of vertex $v_{j_1}^1$. Then, for any layer $q + 1 < i \leq q + 1 + \alpha$, we jump the token at layer i of P to vertex $h_{j_{\mu(i)}}^{\mu(i)}$ of Γ_\star . For every $i > q + 1 + \alpha$, we jump the i th vertex of P to vertex v_{j_κ} (since we assume that H ends with a set that maps to V_κ). The fact that we maintain an st -path after every token jump follows from Observation 6 combined with the fact that vertices of P are connected to all vertices of the preceding layer of Γ_\star .

Once we have reached a properly colored st -path P_1 fully contained in Γ_\star (in exactly $10\kappa^2$ steps), we can use a similar strategy to reach a properly colored st -path P_2 fully contained in Γ_{1, j_1} . More formally, we move by increasing order all the tokens of P_1 in such a way the i -th vertex of P_2 is a copy of the $(i + 1)$ -th vertex of P_1 . Note that it is well-defined since, for every i such that $\mu(i) = 1$, the vertex h_{j_1} belongs to P_2 . Observe that during that transformation the vertices “shift one layer to the left”. We then use a similar transformation to transform P_2 into a path P_3 fully contained in $\Gamma_{1, \star}$. We use $20\kappa^2$ steps from P_1 to P_3 .

We repeat this procedure for every $2 \leq i \leq \kappa$ to transform the path in $\Gamma_{i-1, \star}$ into a path in $\Gamma_{i, \star}$ in $20\kappa^2$ jumps. Then we need an extra $10\kappa^2$ steps to go from $\Gamma_{\kappa, \star}$ to Q (using the converse of the transformation from P to Γ_\star). Hence, the length of the reconfiguration sequence is exactly $20(\kappa^3 + \kappa^2)$. ◀

In order to prove the other direction, we first establish some useful properties of our construction. We let $\Gamma_\star = \Gamma_{0,0}$ and $\Gamma_{i, \star} = \Gamma_{i, n+1}$. We say $\Gamma_{i,j}$ comes before or above $\Gamma_{i', j'}$ whenever $i < i'$ or $i = i'$ and $j < j'$ (we also assume that P appears first and Q appears last, i.e., $P = \Gamma_{-1, -1}$ and $Q = \Gamma_{n+1, n+1}$). We say that two consecutive internal vertices v_p and v_{p+1} of an st -path P are *siblings* if they belong to the same graph $\Gamma_{i,j}$ (that is they belong to the same row in the representation of Figure 2). Otherwise, we say v_p is *above* (or *below*) v_{p+1} if the graph of v_p is above (below) that of v_{p+1} (that is v_p is in the row above or below v_{p+1} in the representation of Figure 2).

► **Lemma 8** (\star). *Let P be a shortest path from s to t in G' . Let v_p denote the p th internal vertex of P . Then:*

- For every p , v_p is a vertex of the p th layer of G' .
- For every two consecutive internal vertices of P , v_p and v_{p+1} , either v_p and v_{p+1} are siblings or v_p is below v_{p+1} .
- For every p , if v_p belongs to $\Gamma_{i,j}$ then no vertex $v_{p'}$ with $p' \geq p$ is below v_p .
- For every p , if v_p belongs to $\Gamma_{i,j}$ then v_{p-1} is either in $\Gamma_{i,j}$ or $\Gamma_{i, n+1}$ and v_{p+1} is either in $\Gamma_{i,j}$ or $\Gamma_{i-1, n+1}$.

Our next result states that the sequence described in Lemma 7 is best possible.

► **Lemma 9** (\star). *Any reconfiguration sequence from P to Q requires at least $20(\kappa^3 + \kappa^2)$ token moves. Moreover, if there exists a reconfiguration sequence from P to Q then there exists one of length exactly $20(\kappa^3 + \kappa^2)$.*

Given Lemma 8 and Lemma 9, it is easy to see that a shortest reconfiguration from P to Q in G' must be *monotone*, i.e., tokens always move towards Q and every path in the reconfiguration sequence consists of a sequence of vertices (ordered from s to t) whose distance from Q monotonically increases.

► **Lemma 10** (\star). *Assume that there exists a reconfiguration sequence σ from P to Q in G' . For $i \in [\kappa]$, let $\mu^{-1}(i) = \{H_{j_1}, H_{j_2}, \dots\}$ denote the H -layers (layers that belong to H) in Γ_\star that map to V_i . Then:*

- *For every two consecutive sets H_j and H_{j+1} in Γ_\star there exists at least one st -path P' in the sequence σ such that P' contains one vertex in both H_j and H_{j+1} .*
- *If σ is a shortest sequence then the intersection of $\bigcup_{P' \in \sigma} V(P')$ with $\bigcup_{H_j \in \mu^{-1}(i)} V(H_j)$ includes only vertices that map to the same vertex of V_i . In other words, for any two vertices w and w' in $W = \bigcup_{P' \in \sigma} V(P') \cap \bigcup_{H_j \in \mu^{-1}(i)} V(H_j)$, we have $\pi(w) = \pi(w')$.*

We now have all the ingredients to finish the proof.

► **Lemma 11.** *If (G', s, t, P, Q) is a yes-instance of SHORTEST PATH RECONFIGURATION then (G, κ) is a yes-instance of (REGULAR) MULTICOLORED CLIQUE.*

Proof. Let (G', s, t, P, Q) be a yes-instance and let σ be a shortest reconfiguration sequence from P to Q . For $i \in [\kappa]$ and $P' \in \sigma$, let $W_i = \bigcup_{P' \in \sigma} V(P') \cap \bigcup_{H_j \in \mu^{-1}(i)} V(H_j)$. Moreover, let $\pi(W_i) = \{\pi(w) \mid w \in W_i\}$. By Lemma 10, we have $|\pi(W_i)| = 1$ and we denote the vertex by $v_{j_i}^i$. Consider the κ vertices $\{v_{j_1}^1, \dots, v_{j_i}^i, \dots, v_{j_\kappa}^\kappa\}$. The fact that those vertices must form a multicolored clique in G again follows from Lemma 10; as every pair must appear consecutively in two H -layers of Γ_\star and some path of σ must intersect with both. ◀

► **Corollary 12** (\star). *SPR is $W[1]$ -hard parameterized by k and SSPR is $W[1]$ -hard parameterized by $k + \ell$ under both the token jumping and the token sliding model.*

3 FPT algorithms

First, we observe that both SPR and SSPR are easily shown to be fixed-parameter tractable when parameterized by $k + \Delta(G)$, where $\Delta(G)$ denotes the maximum degree of G ; by only retaining vertices that belong to some shortest st -path one can easily bound the size of the graph since the i -th layer, consisting of all the vertices at distance exactly i from s , will contain at most $\Delta(G)^i$ vertices. In the remainder of this section, we investigate the complexity of the problem further (and for different parameters) in order to identify the boundary between tractability and intractability. As a warm-up, let us first prove that the following holds:

► **Lemma 13.** *SSPR is FPT parameterized by $k + \ell$ on nowhere-dense classes of graphs for both the sliding and the jumping models.*

Proof. The proof easily follows from the fact that FO-model checking is FPT on nowhere dense classes of graphs [11]. Such an argument has already been used in various proofs for reconfiguration problems, see e.g., [6].

For every $i \leq k$ and $j \leq \ell$, let us create a variable $x_{i,j}$ that represents the i -th vertex of the path at the j -th step of the reconfiguration sequence. Let us prove that we can formulate the existence of a reconfiguration sequence of length ℓ between P and Q as a FO-formula on the set of variables $x_{i,j}$. First we set $x_{i,1} = p_i$ where p_i is the i -th vertex of the path P . Similarly $x_{i,\ell} = q_i$ where q_i is the i -th vertex of the path Q . We now need to ensure that at every step $j \leq \ell$, the set of variables $x_{i,1}, \dots, x_{i,\ell}$ is a path of G , that is, for every $i \leq k - 1$ and every $j \leq \ell$, $x_{i,j}x_{i+1,j}$ is an edge.

23:12 Parameterized Shortest Path Reconfiguration

We further want to ensure that if one vertex is modified between the j -th path and the $(j + 1)$ -th path then all the other vertices are the same. That is, for every $i, i' \leq k$ and $j \leq \ell - 1$, we have $(x_{i,j} \neq x_{i,j+1} \Rightarrow (x_{i',j} = x_{i',j+1}))$. If we want a reconfiguration sequence with the token sliding rule, we have to add the following constraint: for every $i \leq k, j \leq \ell - 1$, $x_{i,j} = x_{i,j+1}$ or $x_{i,j}x_{i,j+1}$ is an edge. Finally, we add the constraints $x_{1,j} = s$ and $x_{k,j} = t$ for every $j \leq \ell$.

Let us denote by ϕ the resulting formula. Let us prove that there exists a reconfiguration sequence from P to Q of length at most ℓ if and only if ϕ is satisfiable. If there exists a reconfiguration sequence $P_1 = P, \dots, P_r = Q$ with $r \leq \ell$ then we simply have to set $x_{i,j}$ to be the i -th vertex of P_j and $x_{i,j'} = q_i$ for every $j' \geq r$ in order to satisfy all the constraints.

Conversely, assume that there exists an assignment of the variables that satisfies all the constraints. Let us denote by P_j the set of ordered vertices $x_{i,j}$ for $1 \leq i \leq k$. Note that, by hypothesis, P_j is an st -path for every j . Moreover, by definition P_j and P_{j+1} differ on at most one vertex and $P_1 = P$ and $P_\ell = Q$. By removing consecutive paths that are the same we obtain a reconfiguration sequence from P to Q , which completes the proof. ◀

Let us now generalize the previous result and prove that the following holds:

► **Theorem 14.** *SSPR is FPT parameterized by ℓ on nowhere dense classes of graphs for both the sliding and the jumping models.*

Proof. The idea of the proof consists of proving that there exists an equivalent instance where the distance between s and t is bounded by a function of ℓ . The conclusion then directly follows from Lemma 13. To do so, we will prove that we can bound (by a function of ℓ) the set of indices i on which there is a relevant modification on the i -th vertex of the path at some step of the reconfiguration sequence. We will then prove that we can “forget” the vertices which are not in these positions by reducing the length of the shortest paths.

Let (G, s, t, P, Q, ℓ) be an instance of SSPR. Let us denote by S the set of positions on which P and Q differ. Note that if $|S| > \ell$ then we can immediately return false since more than ℓ steps are needed to transform P into Q . So we can assume that $|S| \leq \ell$ in the rest of the proof.

▷ **Claim 15.** If there is a reconfiguration sequence from P to Q of length at most ℓ then there is a reconfiguration sequence from P to Q that only modifies vertices whose indices are at distance at most ℓ from an index of S .

Proof. Let \mathcal{R} be a reconfiguration sequence from P to Q of length at most ℓ . At each step, there is exactly one position where a vertex is modified. Let us denote by R that set of positions where a vertex is modified. We have $|R| \leq \ell$. A *component* R' of R is a maximal subset of R containing consecutive integers. Every component R' has a minimum and a maximum value (that might be equal). We say that a component is *important* if it contains a vertex of S and *useless* otherwise.

We claim that if there is a useless component R' , removing from \mathcal{R} all the modifications at position c for every $c \in R'$ leaves a reconfiguration sequence from P to Q . Indeed, let us denote by \mathcal{R}' the resulting reconfiguration sequence. First note that since R' is a useless component, the final shortest path is still Q (we cancel modifications on positions where P and Q were identical). Assume now, for a contradiction, that at some step of the reconfiguration sequence in \mathcal{R}' , the set of vertices P_i is not a shortest st -path. Let us denote by u, v the consecutive vertices of P_i that are not adjacent. Since the path is only modified at positions of indices of R' , either the index of u or v is in R' . Moreover, both of them are not in R' since by definition of \mathcal{R}' all the vertices of indices in R' remain the same all

along the reconfiguration sequence and the initial set of vertices is indeed a path. So we can assume by symmetry that the position of u is the index just before the minimum value of R' and v is the minimum value of R' . Since the vertex v belongs to all the sets in the reconfiguration sequence \mathcal{R}' , it means that u has been modified. But then u should be added in the component R' of v , a contradiction.

Thus, if there is a reconfiguration sequence from P to Q of length ℓ , there is one with no useless component. But the width of a component is at most ℓ since only ℓ vertices are modified in a reconfiguration sequence. So if there is a reconfiguration sequence, there is one that only moves tokens on vertices whose indices are at distance at most ℓ from an index of S , as claimed. \triangleleft

Let $X(i, s)$ be the set of vertices at distance exactly i from s in G . Let I_S be the set of indices at distance at most ℓ from an index of S . Note that I_S has size at most $2\ell \cdot |S|$. An empty interval for I_S is an interval maximal by inclusion in $\{0, \dots, d(s, t)\} \setminus I_S$. Note that I_S has at most $|S|$ empty intervals. We create the graph G' from G as follows:

- G' contains s, t and, for every $i \in I_S$, G' contains all the vertices of $X(i, s)$.
- For all the integers $i \notin I_S$ but at distance one from an integer of I_S , G' contains the vertex at position i in P (and Q).
- There is an edge between x and y if xy is an edge of G , or if x, y are the unique two vertices of G whose positions are in the same empty interval for I_S ⁴.

Let us denote by P' and Q' in G' the set $P \cap V(G')$ and $Q \cap V(G')$. One can easily remark that P' and Q' are shortest st -paths in G' .

\triangleright **Claim 16.** There is a reconfiguration sequence from P to Q in G if and only if there is a reconfiguration sequence from P' to Q' in G' .

Proof. The proof follows from the fact that we can assume that a transformation from P to Q of length at most ℓ in G only modifies vertices whose indices are at distance at most ℓ from an index of S . All those vertices are in G' and all the vertices of G' that contain non-movable tokens are unique at their corresponding distance from s (hence cannot move in G'). \triangleleft

One can remark that the distance between s and t in G' is at most $4\ell^2$. So by Lemma 13, we can decide in FPT-time in ℓ if there is a reconfiguration sequence from P' to Q' in G' , which completes the proof. \blacktriangleleft

References

- 1 John Asplund, Kossi D. Edoh, Ruth Haas, Yulia Hristova, Beth Novick, and Brett Werner. Reconfiguration graphs of shortest paths. *Discret. Math.*, 341(10):2938–2948, 2018. doi:10.1016/J.DISC.2018.07.007.
- 2 Valentin Bartier, Nicolas Bousquet, and Amer E. Mouawad. Galactic token sliding. *J. Comput. Syst. Sci.*, 136:220–248, 2023. doi:10.1016/J.JCSS.2023.03.008.
- 3 Paul S. Bonsma. The complexity of rerouting shortest paths. *Theor. Comput. Sci.*, 510:1–12, 2013. doi:10.1016/j.tcs.2013.09.012.
- 4 Paul S. Bonsma. Rerouting shortest paths in planar graphs. *Discret. Appl. Math.*, 231:95–112, 2017. doi:10.1016/j.dam.2016.05.024.

⁴ Informally, we link the vertex of P just after an interval of I_S with the vertex of P just before the beginning of the next interval of I_S .

- 5 Nicolas Bousquet, Felix Hommelsheim, Yusuke Kobayashi, Moritz Mühlenthaler, and Akira Suzuki. Feedback vertex set reconfiguration in planar graphs. *Theor. Comput. Sci.*, 979:114188, 2023. doi:10.1016/j.tcs.2023.114188.
- 6 Nicolas Bousquet, Amer E. Mouawad, Naomi Nishimura, and Sebastian Siebertz. A survey on the parameterized complexity of the independent set and (connected) dominating set reconfiguration problems. *CoRR*, abs/2204.10526, 2022. doi:10.48550/arXiv.2204.10526.
- 7 Erik D. Demaine, David Eppstein, Adam Hesterberg, Kshitij Jain, Anna Lubiw, Ryuhei Uehara, and Yushi Uno. Reconfiguring undirected paths. In *Algorithms and Data Structures - 16th International Symposium, WADS 2019, Edmonton, AB, Canada, August 5-7, 2019, Proceedings*, pages 353–365, 2019. doi:10.1007/978-3-030-24766-9_26.
- 8 Naoki Domon, Akira Suzuki, Yuma Tamura, and Xiao Zhou. The shortest path reconfiguration problem based on relaxation of reconfiguration rules. In Ryuhei Uehara, Katsuhisa Yamanaka, and Hsu-Chun Yen, editors, *WALCOM: Algorithms and Computation - 18th International Conference and Workshops on Algorithms and Computation, WALCOM 2024, Kanazawa, Japan, March 18-20, 2024, Proceedings*, volume 14549 of *Lecture Notes in Computer Science*, pages 227–241. Springer, 2024. doi:10.1007/978-981-97-0566-5_17.
- 9 Kshitij Gajjar, Agastya Vibhuti Jha, Manish Kumar, and Abhiruk Lahiri. Reconfiguring shortest paths in graphs. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Virtual Event, February 22 - March 1, 2022*, pages 9758–9766, 2022. doi:10.1609/aaai.v36i9.21211.
- 10 Sevag Gharibian and Jamie Sikora. Ground state connectivity of local hamiltonians. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 617–628. Springer, 2015. doi:10.1007/978-3-662-47672-7_50.
- 11 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 89–98. ACM, 2014. doi:10.1145/2591796.2591851.
- 12 Siddharth Gupta, Guy Sa’ar, and Meirav Zehavi. The parameterized complexity of motion planning for snake-like robots. *J. Artif. Intell. Res.*, 69:191–229, 2020. doi:10.1613/JAIR.1.11864.
- 13 Takehiro Ito, Erik D. Demaine, Nicholas J. A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theor. Comput. Sci.*, 412(12-14):1054–1065, 2011. doi:10.1016/j.tcs.2010.12.005.
- 14 Takehiro Ito, Marcin Kaminski, and Erik D. Demaine. Reconfiguration of list edge-colorings in a graph. *Discret. Appl. Math.*, 160(15):2199–2207, 2012. doi:10.1016/j.dam.2012.05.014.
- 15 Anna Lubiw and Vinayak Pathak. Flip distance between two triangulations of a point set is NP-complete. *Comput. Geom.*, 49:17–23, 2015. doi:10.1016/j.comgeo.2014.11.001.
- 16 Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):52, 2018. doi:10.3390/a11040052.
- 17 Krzysztof Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *Journal of Computer and System Sciences*, 67(4):757–771, 2003. doi:10.1016/S0022-0000(03)00078-3.
- 18 Jan van den Heuvel. The complexity of change. In Simon R. Blackburn, Stefanie Gerke, and Mark Wildon, editors, *Surveys in Combinatorics 2013*, volume 409 of *London Mathematical Society Lecture Note Series*, pages 127–160. Cambridge University Press, 2013. doi:10.1017/CB09781139506748.005.
- 19 Marcin Wrochna. Reconfiguration in bounded bandwidth and tree-depth. *J. Comput. Syst. Sci.*, 93:1–10, 2018. doi:10.1016/J.JCSS.2017.11.003.

Roman Hitting Functions

Henning Fernau   

Fachbereich IV, Informatikwissenschaften, Universität Trier, Germany

Kevin Mann  

Fachbereich IV, Informatikwissenschaften, Universität Trier, Germany

Abstract

Roman domination formalizes a military strategy going back to Constantine the Great. Here, armies are placed in different regions. A region is secured if there is at least one army in this region or there are two armies in one neighbored region. This simple strategy can be easily translated into a graph-theoretic question. The placement of armies is described by a function which maps each vertex to 0, 1 or 2. Such a function is called *Roman dominating* if each vertex with value 0 has a neighbor with value 2.

Roman domination is one of few examples where the related (so-called) extension problem is polynomial-time solvable even if the original decision problem is NP-complete. This is interesting as it allows to establish polynomial-delay enumeration algorithms for listing minimal Roman dominating functions, while it is open for more than four decades if all minimal dominating sets of a graph or (equivalently) if all hitting sets of a hypergraph can be enumerated with polynomial delay, or even in output-polynomial time. To find the reason why this is the case, we combine the idea of hitting set with the idea of Roman domination. We hence obtain and study a new problem, called ROMAN HITTING FUNCTION, generalizing ROMAN DOMINATION towards hypergraphs. This allows us to delineate the frontier of polynomial-delay enumerability.

Our main focus is on the extension version of this problem, as this was the key problem when coping with Roman domination functions. While doing this, we find some conditions under which the EXTENSION ROMAN HITTING FUNCTION problem is NP-complete. We then use parameterized complexity as a tool to get a better understanding of why EXTENSION ROMAN HITTING FUNCTION behaves in this way. From an alternative perspective, we can say that we use the idea of parameterization to study the question what makes certain enumeration problems that difficult.

Also, we discuss another generalization of EXTENSION ROMAN DOMINATION, where both a lower and an upper bound on the sought minimal Roman domination function is provided. The additional upper bound makes the problem hard (again), and the applied parameterized complexity analysis (only) provides hardness results.

Also from the viewpoint of Parameterized Complexity, the studies on extension problems are quite interesting as they provide more and more examples of parameterized problems complete for $W[3]$, a complexity class with only very few natural members known five years ago.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases enumeration problems, polynomial delay, domination problems, hitting set, Roman domination

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.24

1 Introduction

For more than four decades, the question if all minimal hitting sets can be enumerated with polynomial delay is an open question. This TRANSVERSAL HYPERGRAPH PROBLEM is equivalent to the question if all minimal dominating sets can be enumerated with polynomial delay. From the point of view of applications, it is quite important to find an affirmative answer: no user likes to wait “forever” to see the next solution, or to get to know that no further solution exist. In order to explore this question, the problem of enumerating minimal



© Henning Fernau and Kevin Mann;

licensed under Creative Commons License CC-BY 4.0

19th International Symposium on Parameterized and Exact Computation (IPEC 2024).

Editors: Édouard Bonnet and Paweł Rzażewski; Article No. 24; pp. 24:1–24:15

Leibniz International Proceedings in Informatics



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

dominating sets has been investigated in many special graph classes. Several graph classes have been identified where this enumeration problem can be solved with polynomial delay. In this paper, we approach this problem from a different side. Namely, in [4] it was shown that all minimal Roman dominating functions¹ can be enumerated with polynomial delay. This was a rather surprising finding as in most other complexity aspects, Roman Domination and Dominating Set behave quite the same. More specifically, to mention some of these results:

- ROMAN DOMINATION is NP-complete, even on special graph classes; see [20, 35].
- MINIMUM ROMAN DOMINATION can be approximated up to a logarithmic factor but not any better, unless $P = NP$, confer [1, 35, 34].
- ROMAN DOMINATION under standard parameterization (by an upper-bound k on the number of armies) is complete for $W[2]$; see [22]. However, the dual parameterization puts ROMAN DOMINATION in FPT; see [2, 6, 7].

So, we take this as a basis and try to generalize Roman Domination towards Hitting Set to understand when or where the polynomial delay feature disappears. In passing, we also generalize the enumeration results from [4] considerably. As we will explain in the following, we introduce a generalizations of Roman Domination towards hypergraphs; we can describe the tractability frontier (with respect to polynomial delay) quite accurately. This also adds to the understanding what lets the Roman variation of domination behave that differently from the classical setting when it comes to enumeration.

Apart from quite a number of combinatorial (graph-theoretic) results that have been obtained for Roman domination, nicely surveyed in a 45-page chapter of [27], the decision problem ROMAN DOMINATION has been studied from various aspects. Even though ROMAN DOMINATION and DOMINATING SET behave the same in terms of complexity in a variety of settings this parallelism unexpectedly breaks down for two (mutually related) tasks:

- Can we enumerate all minimal solutions of a given instance with polynomial delay?
- Can we decide, given a certain part of the solution, if there exists a minimal solution that extends the given pre-solution?

The first type of question is also known as an *output-sensitive enumeration* problem. Even the less demanding task to enumerate all minimal dominating sets in output-polynomial time is open. The corresponding enumeration question for Roman dominating functions can be solved with polynomial delay, as proven in [4] and used in [3]. This result is based on another result giving a polynomial-time algorithm for the *extension problem(s)* as described in the second item. The idea is to call an extension test before diving further into branching. This strategy is well-established in the area of enumeration algorithms, dating back to Read and Tarjan [36], but few concrete examples are known; we only refer to the discussion in [33, 37]. This makes EXTENSION ROMAN DOMINATION one of few examples where the extension problem is polynomial-time solvable, while the original problem is NP-complete. For more details on extension problems, we refer to the survey [13].

The simple scientific question that we want to investigate in this paper is “why”: *What causes Roman domination to be feasible with respect to enumeration and extension?* To find out why EXTENSION ROMAN DOMINATION (EXT RD for short) behaves in this peculiar way and what can be seen as a difference to EXTENSION DOMINATING SET (EXT DS for short), we are going to generalize the concept of Roman domination and try to find the borderline of tractability. By this, we refer to the question if minimal hitting sets can be enumerated with polynomial delay. This so-called TRANSVERSAL HYPERGRAPH PROBLEM [21] is open for four decades. It is quite important, as it appears in many application areas, and in particular

¹ These technical notions will be defined below.

in databases, there are quite a number of interesting equivalent problems, or problems that are shown to be *transversal-hard*, as called in [25]; we only mention two recent references and refer to the papers cited therein: [9, 10]. This question is equivalent to several enumeration problems in logic, database theory and also to enumerating minimal dominating sets in graphs, see [17, 21, 24, 29]. Our paper can be read as trying to understand which kind of problems are transversal-hard, and furthermore, to describe situations when polynomial-delay enumeration algorithms exist. Previous research on enumeration algorithms for minimal dominating sets often tried to look into special graph classes where polynomial-delay enumeration could be exhibited, or not; cf. [28, 29] as examples. This approach can be seen as specializing a known (transversal-hard) enumeration problem by studying special graph classes. Our approach is different as we come from a domination-type problem with a known polynomial-delay enumeration algorithm for general graphs and we try to stretch this result by generalization to understand when this enumeration task becomes transversal-hard.

It is well-known that HITTING SET (HS for short) can be viewed as a generalization of DOMINATING SET (DS for short) by modelling a graph by the closed-neighborhood hypergraph. One of the main differences between Dominating Set (in graphs) and Hitting Set is that in the second setting, there is a clear distinction between the objects that can dominate (the vertices of the hypergraph) and the objects that should be dominated (which are the hyperedges, i.e., sets of vertices). Although HS and ROMAN DOMINATION are both established concepts that generalize DS, it seems that there is no combination of both concepts published in the literature. Actually, trying to define such a combination comes with some problems. If we want a HS instance to represent a DS instance by the closed-neighborhood hypergraph, the vertex set of the given graph is the vertex set of the hypergraph, and the set of all closed neighborhoods is the (hyper)edge set. Ignoring twins, this implies a bijection between the universe and the (hyper)edge set. But in general hypergraphs, the number of hyperedges and the number of elements in the universe are independent. Therefore, we have to think about how to interpret the “value one setting” such that it is related to the definition of ROMAN DOMINATION where exactly one army is put on a certain vertex. We suggest modelling this effect in hypergraphs based on the following idea: If a vertex has the value 1 under a Roman dominating function, then it hits only its “own” closed-neighborhood hyperedge. In general hypergraphs, we have to explicitly express how a vertex “owns” a hyperedge. Hence, we need a function, called *correspondence*, which maps a vertex to an incident hyperedge, such that this hyperedge is dominated if the vertex has the value 1. This hypergraph problem is defined more formally in the next section.

2 Definitions and Notation

Throughout this paper, we will freely use standard notions from complexity theory without defining them here. This includes notions from parameterized complexity, concerning FPT and the further lower levels of the W -hierarchy up to $W[3]$, as described in textbooks like [23, 19].

Let \mathbb{N} denote the set of all nonnegative integers (including 0). For $n \in \mathbb{N}$, we will use the notation $[n] := \{1, \dots, n\}$. For a finite set A and some $n \in \mathbb{N}$ with $n \leq |A|$, the cardinality of A , $\binom{A}{n}$ denotes the set of all subsets of A of cardinality n , while 2^A denotes the power set of A . For two sets A, B , B^A denotes the set of all mappings $f : A \rightarrow B$. If $C \subseteq A$, then $f(C) = \{f(x) \mid x \in C\} \subseteq B$. We denote by $\chi_C \in \{0, 1\}^A$ the characteristic function, where $\chi_C(x) = 1$ holds iff $x \in C$. For two functions $f, g \in \mathbb{N}^A$, we write $f \leq g$ iff $f(a) \leq g(a)$ holds for all $a \in A$. Further, we define the *weight* of f by $\omega(f) = \sum_{a \in A} f(a)$.

We focus on not necessarily simple hypergraphs $H = (X, \hat{S} = (s_i)_{i \in I})$ with a finite universe X , also called *vertex set*, and a finite index set I , where for each $i \in I$, $s_i \subseteq X$ is a *hyperedge*². With the set S we denote the set which includes all hyperedges of the family \hat{S} , i.e., $S = \{s_i \mid i \in I\}$. Note that the same hyperedge may appear multiple times in the family \hat{S} . If this is forbidden, we speak of a *simple hypergraph*. For all $x \in X$, define $\mathbf{S}(x) = \{s_i \in S \mid x \in s_i\}$ as the set of hyperedges that is *hit* by the vertex x , hence defining a function $\mathbf{S} : X \rightarrow 2^S$. A set $D \subseteq X$ is a *hitting set* (*hs* for short) iff $\mathbf{S}(D) = S$, where $\mathbf{S}(D) = \bigcup_{x \in D} \mathbf{S}(x)$. Similarly, define $\mathbf{I} : X \rightarrow 2^I$ by $\mathbf{I}(x) = \{i \in I \mid x \in s_i\}$, extending to $A \subseteq X$ by $\mathbf{I}(A) = \bigcup_{x \in A} \mathbf{I}(x)$. We call $\tau : X \rightarrow I$ a *correspondence* if $x \in s_{\tau(x)}$ for all $x \in X$ or if, in other words, $\tau(x) \in \mathbf{I}(x)$ for all $x \in X$.

Consider a (simple undirected) graph $G = (V, E)$ as a hypergraph $G = (V, \hat{E})$, where each hyperedge contains exactly two elements. Now, we call the hyperedges just edges. Talking about simple graphs, we can consider E as the index set. For each vertex $v \in V$, define its neighborhood as $N(v) = \{u \mid \{v, u\} \in E\}$ and the closed neighborhood as $N[v] = \{v\} \cup N(v)$. For vertex sets $U \subseteq V$, we use $N[U] = \bigcup_{v \in U} N[v]$ for the closed neighborhood of U . A *dominating set* (*ds* for short) of a graph $G = (V, E)$ is a set $D \subseteq V$ such that $N[D] = V$. A function $f : V \rightarrow \{0, 1, 2\}$ is a *Roman dominating function* (*Rdf* for short) iff, for each vertex $v \in V$ with $f(v) = 0$, there exists a $u \in N(v)$ with $f(u) = 2$. A *Rdf* f is *minimal* if for each *Rdf* g with $g \leq f$, $f = g$ holds, leading to the following problem.

Problem name: ROMAN DOMINATION (RD)
Given: A graph $G = (V, E)$ and $k \in \mathbb{N}$
Question: Is there a *Rdf* f with $\omega(f) \leq k$?

Problem name: EXTENSION ROMAN DOMINATION (EXT RD)
Given: A graph $G = (V, E)$ and a function $f : V \rightarrow \{0, 1, 2\}$
Question: Is there a minimal *rd* g for G with $f \leq g$?

Somewhat surprisingly, the extension problem in the second box was proven to be polynomial-time solvable in [4]. This implies that minimal *Rdf* can be enumerated with polynomial delay. The basis of this algorithm is a combinatorial characterization of minimal *Rdfs*. To be able to formulate the combinatorial characterization of minimal *Rdf*, we need a further notion. For $D \subseteq V$ and $v \in D$, define the *private neighborhood* of $v \in V$ with respect to D as $P_{G,D}(v) := N[v] \setminus N[D \setminus \{v\}]$ whose elements are the private neighbors of v .

► **Theorem 2.1** ([4]). *Let $G = (V, E)$ be a graph and $f : V \rightarrow \{0, 1, 2\}$ be a function. Abbreviate $G' := G[f^{-1}(0) \cup f^{-1}(2)]$. Then, f is a minimal *Rdf* iff we find:*

1. $N[f^{-1}(2)] \cap f^{-1}(1) = \emptyset$,
2. $\forall v \in f^{-1}(2) : P_{G', f^{-1}(2)}(v) \not\subseteq \{v\}$, also called *privacy condition*, and
3. $f^{-1}(2)$ is a minimal *ds* of G' .

In order to explore why these results were possible, we generalize these notions and problems for hypergraphs in two ways, with a clear focus on the second possibility.

The first one is probably the most natural one, formed in analogy to the notion of a *ds* in a (simple) hypergraph; see [5]. Let $H = (V, E)$ be a (simple) hypergraph, i.e., $E \subseteq 2^V$. Then, $f : V \rightarrow \{0, 1, 2\}$ is a *Rdf* if, for all $v \in V$ with $f(v) = 0$, there is a vertex $u \in V$ with $f(u) = 2$ that is a neighbor of v , i.e., it shares an edge with v , which means, more formally,

² For our proofs, we found this index notation more convenient than a multiset notation.

that there exists some $e \in E$ with $\{u, v\} \subseteq e$. However, as we show next, we can transfer all interesting properties of Roman domination from the graph case to the hypergraph setting by using the same reduction: if $G = (V, E)$ is a simple hypergraph, we can construct a graph $G' = (V, E')$ by setting $\{x, y\} \in E'$ iff there is a hyperedge $e \in E$ with $\{x, y\} \subseteq e$. This construction is also known as the *Gaifman graph* of G . Then, $f : V \rightarrow \{0, 1, 2\}$ is a Rdf of the hypergraph G iff f is a Rdf of the graph G' . Conversely, we just have to interpret a given graph as a simple hypergraph.

Therefore, we propose a further generalization of Roman domination towards hypergraphs that allow us to study why Roman domination shows such a peculiar behavior when it comes to its extension version, as well as concerning enumerating minimal Rdfs:

Let $H = (X, \hat{S} = (s_i)_{i \in I})$ be a hypergraph and $\tau : X \rightarrow I$ be a correspondence. We call a function $f : X \rightarrow \{0, 1, 2\}$ a *Roman hitting function* (Rhf for short) if, for each $i \in I$, there is an $x \in s_i$ with $f(x) = 2$ or if there exists an $x \in X$ with $\tau(x) = i$ and $f(x) = 1$. In these scenarios, we say that x hits i or s_i . For a function $f : X \rightarrow \{0, 1, 2\}$, we define the partition $P(f) = \{f^{-1}(0), f^{-1}(1), f^{-1}(2)\}$. We now define two decision problems related to Rhf.

Problem name: ROMAN HITTING FUNCTION, or RHF for short

Given: A finite set X , a hyperedge family $\hat{S} = (s_i)_{i \in I}$, forming the hypergraph (X, \hat{S}) , a correspondence $\tau : X \rightarrow I$, and $k \in \mathbb{N}$

Question: Is there a Rhf f with $\omega(f) \leq k$?

Problem name: EXTENSION ROMAN HITTING FUNCTION, or EXT RHF for short

Given: A finite set X , a hyperedge family $\hat{S} = (s_i)_{i \in I}$, forming the hypergraph (X, \hat{S}) , a correspondence $\tau : X \rightarrow I$, and $f : X \rightarrow \{0, 1, 2\}$

Question: Is there a minimal Rhf g with $f \leq g$?

To understand in which way this setting generalizes RD, recall that there are alternative ways to specify a graph as a hypergraph; namely, the *closed-neighborhood hypergraph* G_{nb} associated to a graph $G = (V, E)$ can be described as $G_{nb} = (V, (N[v])_{v \in V})$. Clearly, $D \subseteq V$ is a ds iff D is a hs of G_{nb} . As $v \in N[v]$, the identity can be viewed as a correspondence. In this interpretation, $f : V \rightarrow \{0, 1, 2\}$ is a Rdf of G iff it is a Rhf of G_{nb} .

Organization of the Paper and Main Results. In Section 3, we will prove that our optimization problem is NP-complete and, more interestingly, k -RHF is W[2]-complete. Then, we turn our attention to the extension problem. Recall that the algorithmic results in the case of Roman domination were based on some basic combinatorial insights. Following this logic, first we show in Section 4 a combinatorial characterization of minimal Rhf that we can make use of in Section 5 where we prove that EXT RHF with surjective correspondences can be solved in polynomial time. In Section 6, we return to Roman domination and consider a variant of the extension problem where we give both lower and upper bound conditions to the minimal Rdf that we are looking for. In contrast to the original problem (that only provides a lower bound), this two-sided extension problem turns out to be NP-complete as we show. Furthermore, we identify two natural parameters under which bounded-EXT RD is W[3]-complete. In Section 7, we further discuss different parameterization of EXT RHF. Again, we obtain some parameterizations for EXT RHF where the problem is W[3]-complete. To save space, we will mark theorems with (*) if the proof is in the long version of the paper.

3 The Optimization Problem RHF

In this section, we will discuss the (parameterized) complexity of the optimization problem RHF. The probably most natural parameterization for the problem is by an upper bound k on the weight of the Rhf. For our results, we will use the $W[2]$ -completeness of RD (with k as parameter) shown in [22]. The hardness follows, as Rdf can be interpreted as Rhf. For the membership, we construct a split graph, where the clique represents the elements of the universe and the independent set form the hyperedges. Here, the hyperedges are added twice to the independent set if the inverse image of the hyperedge with respect to τ is empty. In this way, it is better to put a neighbor to 2 than to put these two vertices to 1.

► **Theorem 3.1.** (*) *RHF is NP-complete. k -RHF is $W[2]$ -complete.*

The fact that RD is NP-complete even on split graphs was mentioned repeatedly in the literature, for instance, in [16, 30], but to the best of our knowledge, no proof of this fact has been published. We will provide a strengthened assertion in the following. Recall that two vertices u, v in a graph are called *true twins* if $N[u] = N[v]$.

► **Lemma 3.2.** (*) *RD is NP-complete even on true-twin-free split graphs. Likewise, k -RD is $W[2]$ -complete on true-twin-free split graphs.*

The hardness part of the proof of Theorem 3.1 (implicitly) uses the fact that the family of hyperedges could include a hyperedge multiple times as there could be twins (vertices with the same closed neighborhoods) in the original graph. Consider a complete graph $K_n = ([n], E_n)$ with $n \geq 2$ vertices. Since the closed neighborhoods are always equal to $[n]$, if we would use a normal set for the hyperedges instead of a family of hyperedges, then the best solution would be $\chi_{\{v\}}$ for any vertex v . This would even not be a Rdf. Namely, minimal Rdf would be of the form $2 \cdot \chi_{\{v\}}$ for any vertex v , or they would be constant 1. Nevertheless, the following holds, revisiting Lemma 3.2.

► **Corollary 3.3.** *RHF is NP-complete even on simple hypergraphs. Furthermore, k -RHF is $W[2]$ -complete.*

The NP-completeness of the optimization problem also motivates our analysis of the extension problem; it could help speed up an exact branching algorithm for solving this decision problem. In the long version, we also consider approximation complexity for RHF.

4 Combinatorial Properties of Minimal Rhf

In this section, we will prove combinatorial properties of minimal Rhf. This will help us analyze the complexity of EXT RHF.

► **Theorem 4.1.** *Let X be a vertex set, $\hat{S} = (s_i)_{i \in I}$ be a hyperedge family and $\tau : X \rightarrow I$ be a correspondence. Then, a function $f : X \rightarrow \{0, 1, 2\}$ is a minimal Rhf iff the following constraint items hold:*

0. $\forall x, y \in f^{-1}(1) : x \neq y \Rightarrow \tau(x) \neq \tau(y)$,
1. $\forall x \in f^{-1}(1) : s_{\tau(x)} \cap f^{-1}(2) = \emptyset$,
2. $\forall x \in f^{-1}(2) \exists i \in I \setminus \{\tau(x)\} : s_i \cap f^{-1}(2) = \{x\}$, and
3. $f^{-1}(2)$ is a minimal hs on $\{s_i \in S \mid i \in I, \tau^{-1}(i) \cap f^{-1}(1) = \emptyset\}$.

Proof. Let f be a minimal Rhf on X, \hat{S} and τ . Assume there are $x, y \in f^{-1}(1)$ with $x \neq y$ but $\tau(x) = \tau(y) = i$. Define $\tilde{f} = f - \chi_{\{y\}}$. Trivially, $\tilde{f} \leq f$ and $\tilde{f} \neq f$. Since $f^{-1}(2) = \tilde{f}^{-1}(2)$ and $\tau(\tilde{f}^{-1}(1)) = \tau(f^{-1}(1) \setminus \{y\}) = \tau(f^{-1}(1))$ hold, \tilde{f} is a Rhf. Thus, f is not minimal, which is a contradiction.

Now assume there is an $x \in f^{-1}(1)$ with $s_{\tau(x)} \cap f^{-1}(2) \neq \emptyset$. Define $\tilde{f} = f - \chi_{\{x\}}$. Trivially, $\tilde{f} \leq f$, $f \neq \tilde{f}$ and $f^{-1}(2) = \tilde{f}^{-1}(2)$. Hence, $s_{\tau(x)} \in \mathbf{S}(\tilde{f}^{-1}(2))$. Since s_i , for $i \in I \setminus \{\tau(x)\}$, is hit by \tilde{f} in the same way as by f , \tilde{f} is a Rhf. This contradicts the minimality of f .

Assume $f^{-1}(2)$ is not a minimal hs on S' . If there is an $s \in S'$ that is not hit by $f^{-1}(2)$, then f is no Rhf, contradicting our assumption. Hence, we can assume $f^{-1}(2)$ is not minimal. More explicitly we assume there is an $x \in f^{-1}(2)$ such that for each $i \in I \setminus \{\tau(x)\}$, $s_i \cap f^{-1}(2) \neq \{x\}$. Then there is an $x \in f^{-1}(2)$ with, for each $s \in \mathbf{S}'(x) \setminus \{\tau(x)\}$, there exists a $y \in (f^{-1}(2) \setminus \{x\}) \cap s$. Define $\tilde{f} = f - \chi_{\{x\}}$. Let $i \in I$. As $\tau(f^{-1}(1)) \cup \{\tau(x)\} = \tau(\tilde{f}^{-1}(1))$ holds by definition, we only need to consider $i \in I \setminus (\tau(f^{-1}(1)) \cup \tau(x))$. For i with $s_i \in \mathbf{S}(f^{-1}(2)) \setminus \mathbf{S}(x)$, trivially $s_i \cap \tilde{f}^{-1}(2) \neq \emptyset$. If $s_i \in \mathbf{S}(x) \setminus \{\tau(x)\}$, as we mentioned before, there is a $y \in (f^{-1}(2) \setminus \{x\}) \cap s_i = \tilde{f}^{-1}(2) \cap s_i$. Thus, \tilde{f} is a Rhf and f is not minimal, contradicting our assumption. Hence, the four conditions hold.

For the if-part assume f fulfills the constraints. By Constraint 3, for all $i \in I$, either $s_i \cap f^{-1}(2) \neq \emptyset$, or there is an $x \in f^{-1}(1)$ with $\tau(x) = i$. Hence, f is a Rhf. Let $g : X \rightarrow \{0, 1, 2\}$ be a minimal Rhf with $g \leq f$. Thus, $g^{-1}(2) \subseteq f^{-1}(2)$ and $\mathbf{S}(g^{-1}(2)) \subseteq \mathbf{S}(f^{-1}(2))$ hold. Furthermore, $g^{-1}(1) \subseteq f^{-1}(1) \cup f^{-1}(2)$. Since for each $x \in X$, $\{s_{\tau(x)}\} \subseteq \mathbf{S}(x)$, for each $i \in \tau(g^{-1}(1))$, $i \in \tau(f^{-1}(1))$ or $s_i \in \mathbf{S}(f^{-1}(2))$. Let $x \in X$ be an element with $g(x) < f(x)$.

Case 1: $g(x) = 0 < 2 = f(x)$. This implies that, for each $i \in \mathbf{I}(x)$, there exists a $y \in s_i$ with $2 = g(y) \leq f(y) = 2$ or $y \in \tau^{-1}(i) \cap g^{-1}(1) \subset \tau^{-1}(i) \cap (f^{-1}(1) \cup f^{-1}(2))$. This either contradicts Constraint 1 or Constraint 2.

Case 2: $g(x) = 1 < 2 = f(x)$. This case works analogously, somehow simpler. We only need to exclude $i = \tau(x)$.

Case 3: $f(x) = 1$. This implies $g(x) = 0$. Since g is a Rhf, either $s_{\tau(x)} \cap g^{-1}(2)$ is not empty or there exists a $y \in g^{-1}(1)$ with $\tau(x) = \tau(y)$.

Case 3.1: $\tau(x) \cap g^{-1}(2) \neq \emptyset$. As $g^{-1}(2) \subseteq f^{-1}(2)$, this contradicts Constraint 1.

Case 3.2: There is a $y \in g^{-1}(1)$ with $\tau(x) = \tau(y)$. Thus, either there is a $y \in f^{-1}(1) \setminus \{x\}$ with $\tau(x) = \tau(y)$ (this contradicts Constraint 0) or $f(y) = 2$ (this contradicts Constraint 1).

Thus, $g = f$ holds. Therefore, f is minimal. \blacktriangleleft

► **Remark 4.2.** One can compare Theorem 4.1 with Theorem 2.1. For a graph $G = (V, E)$, let $G_{nb} = (V, (N[v])_{v \in V})$ be the closed-neighborhood hypergraph. Here, for each $f : V \rightarrow \{0, 1, 2\}$ and $i \in \{1, 2, 3\}$, f fulfills Constraint i of Theorem 4.1 with respect to G_{nb} iff f fulfills Constraint i of Theorem 2.1 with respect to G .

We call a $f : X \rightarrow \{0, 1, 2\}$ *extensible* on the hypergraph $H = (X, \hat{S})$ with correspondence τ if there is a minimal Rhf g with $f \leq g$. The following two results are basically implied by Theorem 4.1.

► **Lemma 4.3.** (*) Let $H = (X, \hat{S} = (s_i)_{i \in I})$ be a hypergraph with correspondence τ and $f : X \rightarrow \{0, 1, 2\}$ be a function with $x \in f^{-1}(2)$, $y \in f^{-1}(1)$ and $x \in s_{\tau(y)}$. Then, f is extensible iff $f + \chi_{\{y\}}$ is extensible.

► **Lemma 4.4.** (*) Let $H = (X, \hat{S} = (s_i)_{i \in I})$ be a hypergraph with correspondence τ and $f : X \rightarrow \{0, 1, 2\}$ be a function with $x, y \in f^{-1}(1)$, $x \neq y$ and $\tau(x) = \tau(y)$. Then, f is extensible iff $f + \chi_{\{x, y\}}$ is extensible.

► **Theorem 4.5.** Let $H = (X, \hat{S} = (s_i)_{i \in I})$ be a hypergraph with correspondence τ , $\tau : X \rightarrow I$. Let $f : X \rightarrow \{0, 1, 2\}$ be a function such that $x \neq y$ implies $\tau(x) \neq \tau(y)$ for each $x, y \in f^{-1}(1)$. Then, f is extensible iff there exist a set R_2 with $f^{-1}(2) \subseteq R_2 \subseteq f^{-1}(1) \cup f^{-1}(2)$ and a mapping $\rho : R_2 \rightarrow I$, satisfying the following constraints.

1. $\forall x \in R_2 : \rho(x) \neq \tau(x)$.
2. $\forall x \in R_2 : s_{\rho(x)} \cap R_2 = \{x\}$.
3. $\forall x \in f^{-1}(1) \setminus R_2 : s_{\tau(x)} \cap R_2 = \emptyset$.
4. $\forall i \in I$ such that $\tau^{-1}(i) = \emptyset$:
 $s_i \subseteq \left(\bigcup_{x \in f^{-1}(1) \setminus R_2} s_{\tau(x)} \right) \cup \left(\bigcup_{x \in R_2} s_{\rho(x)} \right) \implies s_i \cap R_2 \neq \emptyset$.

Proof. Define $I' := \{i \in I \mid \tau^{-1}(i) = \emptyset\}$. First, we assume that f is extensible. Let $g : X \rightarrow \{0, 1, 2\}$ be a minimal Rhf with $f \leq g$. By Constraint item 3 of Theorem 4.1, $g^{-1}(2)$ is a minimal hs on $\{s_i \in S \mid i \in I, \tau^{-1}(i) \cap f^{-1}(1) = \emptyset\}$ (*). With Theorem 4.1, Constraint 2, this implies that, for each $x \in g^{-1}(2)$, there exists an $\rho(x) \in I \setminus \{\tau(x)\}$ such that $s_{\rho(x)} \cap g^{-1}(2) = \{x\}$. Define $R_2 = (f^{-1}(1) \cup f^{-1}(2)) \cap g^{-1}(2)$. Clearly, $f^{-1}(2) \subseteq R_2 \subseteq f^{-1}(1) \cup f^{-1}(2)$. We have to check the four constraints claimed for R_2 . The first two are even true in a slightly more general fashion by (*). If there would exist a $y \in f^{-1}(1) \setminus R_2 \subseteq g^{-1}(1)$ with $\emptyset \neq s_{\tau(y)} \cap R_2 \subseteq s_{\tau(y)} \cap g^{-1}(2)$, then this would contradict Theorem 4.1, Constraint 2, showing the third constraint of this theorem. We now turn to the fourth and last constraint. Let $i \in I'$. Since $\tau^{-1}(i) = \emptyset$, there has to be a $y \in g^{-1}(2) \cap s_i$. If $y \in R_2$, then the constraint is satisfied. Hence, we can assume that $y \in g^{-1}(2) \setminus R_2 = g^{-1}(2) \cap f^{-1}(0)$. Consider $x \in f^{-1}(1) \setminus R_2$. As $f \leq g$ and $x \notin R_2$, we have $g(x) = 1$. By Constraint 1 of Theorem 4.1, $y \notin s_{\tau(x)}$. If $s_i \subseteq \left(\bigcup_{x \in f^{-1}(1) \setminus R_2} s_{\tau(x)} \right) \cup \left(\bigcup_{x \in R_2} s_{\rho(x)} \right)$ and $s_i \cap R_2 = \emptyset$ hold, then this would contradict $s_{\rho(x)} \cap R_2 = \{x\}$. Therefore, all the constraints hold.

Let now f , R_2 and $\rho : R_2 \rightarrow I$ fulfill the constraints of this theorem. For this part of the proof we will define a hypergraph H' that includes each edge where $\tau(X)$ does not include its index and the edge is not hit, yet. We will show that there is a minimal hs D on H' which does not include any vertex of $s_{\rho(x)}$ for $x \in R_2$ or $s_{\tau(x)}$ for $f^{-1}(1) \setminus R_2$. $R_2 \cup D$ will describe the set of vertices with value 2. We will hit the remaining vertices by assigning the value 1 to some vertices. Therefore, we define the hypergraph $H' = (X', (s'_i)_{i \in I''})$ with

$$I'' := I' \cap \{i \in I \mid s_i \cap R_2 = \emptyset\},$$

$$X' := \left(\bigcup_{i \in I''} s_i \right) \setminus \left(\left(\bigcup_{x \in f^{-1}(1) \setminus R_2} s_{\tau(x)} \right) \cup \left(\bigcup_{x \in R_2} s_{\rho(x)} \right) \right)$$

and $s'_i := s_i \cap X'$. If s'_i is empty for an $i \in I''$, then there would not exist any hs on H' . Therefore, we need to ensure that such an index does not exist. Let $i \in I''$. Hence, $\tau^{-1}(i) = \emptyset$ and $s_i \cap R_2 = \emptyset$. The contraposition of the implication of Constraint 4 implies

$$s_i \not\subseteq \left(\bigcup_{x \in f^{-1}(1) \setminus R_2} s_{\tau(x)} \right) \cup \left(\bigcup_{x \in R_2} s_{\rho(x)} \right).$$

Hence, $s'_i \neq \emptyset$ for each $i \in I''$. Thus, there is a minimal hs D on H' . The construction of H' and D implies that $\tau^{-1}(i) \neq \emptyset$ for each $i \in I \setminus \mathbf{I}(R_2 \cup D)$. For each $i \in I \setminus \mathbf{I}(R_2 \cup D)$, x_i will describe an arbitrary vertex in $\tau^{-1}(i)$, unless there exists an $x_i \in (f^{-1}(1) \setminus R_2) \cap \tau^{-1}(i)$ (by assumption on f , there is at most one such element). In this case, we choose this x_i .

Define $g : X \rightarrow \{0, 1, 2\}$ with $g^{-1}(1) = \{x_i \mid i \in I \setminus \mathbf{I}(R_2 \cup D)\}$ and $g^{-1}(2) = D \cup R_2$. We will now use Theorem 4.1 to show that g is a minimal Rhf. By the construction of $g^{-1}(1)$, Constraints 0 and 1 of Theorem 4.1, are fulfilled. Since $g^{-1}(1)$ hits each edge in $I \setminus \mathbf{I}(D \cup R_2)$, each hyperedge in $\{s_i \mid i \in I, \tau^{-1}(i) \cap g^{-1}(1) = \emptyset\}$ is hit by $g^{-1}(2)$. As D is minimal and $D \cap \left(\bigcup_{x \in R_2} s_{\rho(x)} \right) = \emptyset$, Constraint 2 implies that $D \cup R_2$ also a minimal hs on $\mathbf{I}(D \cup R_2)$. The remaining constraint of Theorem 4.1 follows by the first two constraints of f together with definition of H' and D as H' only contains hyperedges s'_i where $\tau^{-1}(i) = \emptyset$. \blacktriangleleft

We will use Lemma 4.4 and Theorem 4.5 to show a $W[3]$ -membership.

► **Remark 4.6.** Theorem 4.5 already gives an idea why EXT RHF with surjective correspondence (and therefore also EXT RD) runs in polynomial time. Let $H = (X, (s_i)_{i \in I})$ be a hypergraph and $f : X \rightarrow \{0, 1, 2\}$ be a function with the surjective correspondence τ . Hence, we can disregard Constraint 4, as $\tau^{-1}(i) \neq \emptyset$ for each $i \in I$. Then, we could use Lemmata 4.3 and 4.4. We set $R_2 := f^{-1}(2)$. By Constraint 3, we have to add each $x \in f^{-1}(1)$ with $s_{\tau(x)} \cap R_2 \neq \emptyset$ to R_2 . Now we can check if for each $x \in R_2$ there is a $i_x \in I$ that fulfills Constraints 1 and 2. This will be our strategy in the next section.

5 Ext RHF with Surjective Correspondence and Ext RD

In this section, we present a polynomial-time algorithm for EXT RHF instances with a surjective correspondence function τ . At the end of this section, we explain how this algorithm can be viewed as a natural generalization of EXT RD that was studied before in [4].

Algorithm 1 ExtRHF Algorithm.

```

1: procedure EXTRHF SOLVER( $X, \hat{S}, \tau, f$ )
   Input: set  $X$ ,  $\hat{S} := (s_i)_{i \in I}$ ,  $\tau$  correspondence function,  $f : X \rightarrow \{0, 1, 2\}$  with  $\{i \in I \mid \tau^{-1}(i) = \emptyset\} \subseteq \mathbf{I}(f^{-1}(2))$ .
   Output: Is there a minimal Rhf  $g$  with  $f \leq g$ ?
2:    $M_2 := f^{-1}(2)$ ,    $M_1 := f^{-1}(1)$ 
3:   for  $x \in M_1$  do
4:     for  $y \in M_1 \setminus \{x\}$  do
5:       if  $\tau(x) = \tau(y)$  then
6:         Add  $x, y$  to  $M_2$  and delete them in  $M_1$ .
7:         Continue with the next  $x$ .
8:    $M := M_2$  { All  $x \in g^{-1}(2)$  are considered in the following. }
9:   while  $M \neq \emptyset$  do
10:    Choose  $x \in M$ .
11:    for  $y \in \tau^{-1}(\mathbf{I}(x))$  do
12:      if  $y \in M_1$  then Add  $y$  to  $M_2$  and  $M$ . Delete  $y$  in  $M_1$ .
13:    Delete  $x$  from  $M$ .
14:   for  $x \in M_2$  do
15:     if  $\mathbf{I}(x) \setminus \mathbf{I}(M_2 \setminus \{x\}) \subseteq \{\tau(x)\}$  then Return no
16:   for  $i \in I \setminus (\mathbf{I}(M_2) \cup \tau(M_1))$  do
17:     Add one arbitrary element  $x \in \tau^{-1}(i)$  to  $M_1$ .
18:   Return yes  $\{g^{-1}(0) = X \setminus (M_1 \cup M_2), g^{-1}(1) = M_1, g^{-1}(2) = M_2\}$ 

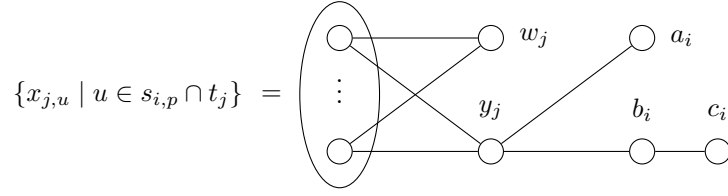
```

► **Theorem 5.1.** (*) *Algorithm 1 solves EXT RHF for instances (X, \hat{S}, τ, f) satisfying $\{i \in I \mid \tau^{-1}(i) = \emptyset\} \subseteq \mathbf{I}(f^{-1}(2))$ in polynomial time.*

A special case of this theorem entails: EXT RHF with surjective τ is polynomial-time solvable. In the long version, we discuss the connections between Algorithm 1 and Algorithm 1 in [4].

6 Bounded Extension Roman Domination

In this section, we will discuss a two-sided bounded version of EXTENSION ROMAN DOMINATION which was also suggested by a colleague of ours.



■ **Figure 1** Construction for Theorem 6.1, for $i \in [k]$, $p \in [\ell_i]$ and $[j \in \ell_T]$.

Problem name: Bounded EXTENSION ROMAN DOMINATION (bounded-EXT RD)

Given: A graph $G = (V, E)$ and functions $f, h : V \rightarrow \{0, 1, 2\}$.

Question: Is there a minimal Rdf $g : V \rightarrow \{0, 1, 2\}$ with $f \leq g \leq h$?

We show in Corollary 6.4 that bounded-EXT RD is NP-complete. Thus, we look for FPT-algorithms. One natural parameterization for this problem could be $\omega(2-h)$, because for $2-h=0$, we are back to EXT RD as a special case, which is known to be solvable in polynomial time. Hence, this parameterization can be viewed as a “distance-from-triviality” parameter [26]. However, as we will prove in Theorem 6.1, this parameterization strategy fails. We employ the well-known W[3]-completeness of MULTICOLORED INDEPENDENT FAMILY (MULTINDFAM), parameterized by k , in the reduction presented in the proof of Theorem 6.1.

Problem name: MULTICOLORED INDEPENDENT FAMILY (MULTINDFAM)

Given: A $(k+1)$ -tuple (S_1, \dots, S_k, T) of subsets of 2^U on the common universe U , i.e., $(U, S_1), \dots, (U, S_k), (U, T)$ are $k+1$ many simple hypergraphs.

Question: Are there hyperedges $s_1 \in S_1, \dots, s_k \in S_k$ such that no $t \in T$ is a subset of $\bigcup_{i=1}^k s_i \subseteq U$?

In that theorem, we actually discuss a slightly different parameterization, namely $\kappa_{h^{-1}(0)}(G, f, h) := |h^{-1}(0)|$.

► **Theorem 6.1.** $\kappa_{h^{-1}(0)}$ -bounded-EXT RD is W[3]-hard even on bipartite graphs.

Proof. Let $k \in \mathbb{N}$ and (S_1, \dots, S_k, T) be a $(k+1)$ -tuple of subsets of 2^U with a common universe U . To simplify the notation, let $T = \{t_1, \dots, t_{\ell_T}\}$ and $S_i = \{s_{i,1}, \dots, s_{i,\ell_i}\}$ for $i \in [k]$. Define $X_j := \{x_{j,u} \mid u \in t_j\}$ for $j \in [\ell_T]$, $Y_i := \{y_{i,1}, \dots, y_{i,\ell_i}\}$ and $G = (V, E)$ with

$$V := \left(\bigcup_{i=1}^k \{a_i, b_i, c_i\} \cup Y_i \right) \cup \left(\bigcup_{j=1}^{\ell_T} \{w_j\} \cup X_j \right),$$

$$E := \{ \{a_i, y_{i,p}\}, \{b_i, y_{i,p}\}, \{b_i, c_i\} \mid i \in [k], p \in [\ell_i] \} \cup \{ \{w_j, x_{j,u}\} \mid j \in [\ell_T], u \in t_j \}$$

$$\cup \{ \{y_{i,p}, x_{j,u}\} \mid i \in [k], p \in [\ell_i], j \in [\ell_T], u \in s_{i,p} \cap t_j \}.$$

Clearly, G is bipartite, as $V = A \cup B$ decomposes V into two disjoint independent sets, with $A = \left(\bigcup_{i=1}^k \{a_i, b_i\} \right) \cup \left(\bigcup_{j=1}^{\ell_T} X_j \right)$ and $B = \left(\bigcup_{i=1}^k \{c_i\} \cup Y_i \right) \cup \left(\bigcup_{j=1}^{\ell_T} \{w_j\} \right)$. Furthermore, we need the maps $f, h \in \{0, 1, 2\}^V$ with $f = 2\chi_{\{w_1, \dots, w_{\ell_T}\} \cup \{b_1, \dots, b_k\}}$ and $h = 2(1 - \chi_{\{a_1, \dots, a_k\}})$.

► **Claim 6.2.** (*) S_1, \dots, S_k, T is a yes-instance of the MULTINDFAM problem iff there exists a minimal Rdf g on G with $f \leq g \leq h$.

Since $k = |\{a_1, \dots, a_k\}| = |h^{-1}(0)|$, this is an FPT-reduction. ◀

Since h maps no vertex to 1 in the reduction presented in the proof of Theorem 6.1, bounded-EXT RD is W[3]-hard, parameterized by $\kappa_{2-h}(G, f, h) := \sum_{v \in V} (2 - h(v))$. Namely, in the construction of Theorem 6.1, $\kappa_{2-h}(G, f, h) = 2 \cdot \kappa_{|h^{-1}(0)|}(G, f, h)$.

Another parameterization could be $\omega(f)$: If $\omega(f) = 0$, there is a minimal Rdf $g : V \rightarrow \{0, 1, 2\}$ with $f \leq g \leq h$ iff h is a Rdf (this can be checked in polynomial time). If there is such a g , then h is also a Rdf. If h is a Rdf, then we can decrease the value of the vertices until we can no longer decrease the value of any vertices without losing the Rdf property. To understand the complexity of this parameter, we need the following extension version of HS.

Problem name: EXTENSION HITTING SET (EXT HS)
Given: A simple hypergraph $H = (X, S)$, $S \subseteq 2^X$, and a set $U \subseteq X$.
Question: Is there a minimal hs $T \subseteq X$ with $U \subseteq T$?

In [8], it was proven that EXT HS is W[3]-complete when parameterized by $|U|$. We use this result in the proof of the following theorem.

► **Theorem 6.3.** (*) $\omega(f)$ -bounded-EXT RD is W[3]-hard on split graphs.

Since the reductions in this section are polynomial-time reductions and since membership in NP is easily seen using guess-and-check, we can conclude:

► **Corollary 6.4.** bounded-EXT RD is NP-complete even on split graphs or bipartite graphs.

We will make use of the W[3]-hardness in the next section, when we turn to discuss the complexity of EXT RHF. The reductions provided there will also show that bounded-EXT RD is W[3]-complete for some parameterizations.

Finally, let us mention that, in any given bounded-EXT RD instance (G, f, h) , we can always assume (1) $f \leq h$ and, moreover, (2) $h(v) = 0$ implies $h(u) = 2$ for some $u \in N(v)$. Otherwise, there cannot exist a Rdf g with $f \leq g \leq h$. Both conditions are easy to check.

7 Complexity of Ext RHF

In this section, we will show that there are instances of EXT RHF which are W[3]-complete, considering their different parameterizations. For the W[3]-membership, we make again use of MULTINDFAM, as there is no further W[3]-complete problems that we find suitable for a reduction. Unfortunately, this reduction is quite technical.

► **Theorem 7.1.** $\omega(f)$ -EXT RHF is in W[3].

Proof. Let $H = (X, \hat{S} = (s_i)_{i \in I})$ be a hypergraph with correspondence $\tau : X \rightarrow I$ and let $f : X \rightarrow \{0, 1, 2\}$ be some function, comprising an instance of $\omega(f)$ -EXT RHF. (*) We can assume that there are not two elements $x, y \in X$ such that $f(x) = f(y) = 1$ with $\tau(x) = \tau(y)$ or $f(x) = 2, f(y) = 1$ with $x \in s_{\tau(y)}$. Otherwise, we could use Corollaries 4.3 and 4.4.

We will construct an equivalent MULTINDFAM instance next. We define its universe as $U := X \cup \{r_{i,x}, x' \mid x \in f^{-1}(\{1, 2\}), i \in I\} \cup \{\tau_x \mid x \in f^{-1}(1)\}$. For the construction of the hypergraphs, we need to define some additional (auxiliary) sets:

- For $x \in f^{-1}(\{1, 2\})$, $i \in I(x)$ abbreviate $\tilde{s}_{x,i} := s_i \cup \{r_{i,x}, x'\}$.
- Define $t_i := s_i \cup \{\tau_x \mid x \in f^{-1}(1) \cap s_i\}$ for $i \in I$ with $\emptyset = \tau^{-1}(i) = s_i \cap f^{-1}(2)$.
- For each $x \in f^{-1}(2)$, let $S_x := \{\tilde{s}_{x,i} \mid \tau(x) \neq i\}$ and for each $x \in f^{-1}(1)$, let $S_x := \{s_{\tau(x)} \cup \{\tau_x\}\} \cup \{\tilde{s}_{x,i} \mid \tau(x) \neq i\}$.

24:12 Roman Hitting Functions

■ Furthermore, we need the target set $T = T' \cup T''$, where

$$\begin{aligned} T' &:= \{t_i \mid i \in I \wedge \tau^{-1}(i) = \emptyset \wedge s_i \cap f^{-1}(2) = \emptyset\} \quad \text{and} \\ T'' &:= \left\{ \{r_{i,x}, y'\} \mid i \in I \wedge \{x, y\} \subseteq (s_i \cap f^{-1}(\{1, 2\})) \wedge x \neq y \right\} \\ &\quad \cup \left\{ \{\tau_x, y'\} \mid x \in g^{-1}(1) \wedge y \in s_{\tau(x)} \wedge x \neq y \right\}. \end{aligned}$$

Now we will explain the idea of each element. It is important to keep in mind that we want to use Theorem 4.5: If x' is in a chosen hyperedge, then we assign the value 2 to x in the minimal Rhf. The element $r_{i,x}$ gives us information about the mapping ρ . $r_{i,x}$ is in one of the chosen edges iff $\rho(x) = i$ holds. Therefore, the sets $\{r_{i,x}, y'\}$ verify the Constraint 2 of Theorem 4.5. τ_x will only be in a set we chose if we assign the value 1 to x . Hence, Constraint 3 will be checked by the sets $\{\tau_x, y'\}$. The sets in T' correspond to the sets which we consider in Constraint 4. This is also the reason why τ_x is included in t_i . Since there exists a τ_x , $f(x) = 1$. If $x \in R_2$, $s_i \cap R_2 \neq \emptyset$. In the MULTINDFAM instance, this corresponds to: τ_x will not be in our sets, which implies that t_i will not be covered completely.

▷ **Claim 7.2.** (*) (H, τ, f) is a yes-instance of EXT RHF iff $(U, (S_x)_{x \in f^{-1}(\{1, 2\})}, T)$ is a yes-instance of MULTINDFAM.

As $|U| \leq |X| \cdot (|I| + 4)$, $|T| \leq |I| + |X|^2 \cdot (|I| + 1)$ and $|S_x| \leq |I|$, the MULTINDFAM can be constructed in polynomial time. Furthermore, the parameter of the constructed instance of MULTINDFAM is $|f^{-1}(1) \cup f^{-1}(2)| \leq \omega(f)$. Hence, EXT RHF belongs to $W[3]$. ◀

As mentioned in the previous proof, the described reduction is also a polynomial-time reduction. Hence, EXT RHF is a member of NP, but this is also observed by the guess-and-check characterization of NP. For the hardness results, we will use bounded-EXT RD.

► **Theorem 7.3.** $\omega(f)$ -EXT RHF is $W[3]$ -hard even if the correspondence function is injective. Furthermore, $\omega(f)$ -bounded-EXT RD is $W[3]$ -complete.

Proof. We will make use of Theorem 6.3, reducing from bounded-EXT RD. Let (G, f, h) be a instance of the bounded-EXT RD, with $G = (V, E)$. We can assume (1) $f \leq h$ and, moreover, (2) $h(v) = 0$ implies $h(u) = 2$ for some $u \in N(v)$. We parameterize by $\omega(f)$. For $v \in X := V \setminus h^{-1}(0)$, define $T_v := (N(v) \setminus h^{-1}(\{0, 1\})) \cup \{v\}$, and for $v \in h^{-1}(0)$, define $T_v := (N(v) \setminus h^{-1}(\{0, 1\}))$. Further, we set $\hat{S} := (T_v)_{v \in V}$ and we define τ as the correspondence satisfying $\tau(v) = v$ and we let $\bar{f} : X \rightarrow \{0, 1, 2\}$, $v \mapsto f(v)$, i.e., $\bar{f} = f|_X$. Let $H = (X, \hat{S})$. Then, (H, τ, \bar{f}) describes an instance of EXT RHF. The parameter is $\omega(\bar{f})$ for this instance. As $f \leq h$, $h(v) = 0$ implies $f(v) = 0$. Thus, $\omega(f) = \omega(\bar{f})$, so that the parameter value does not change when moving from the bounded-EXT RD instance to the $\omega(f)$ -EXT RHF instance. Clearly, the described construction can be carried out in polynomial time. Trivially, τ is injective. What remains to be shown is the following claim.

▷ **Claim 7.4.** (*) (G, f, h) is a yes-instance of bounded-EXT RD iff (H, τ, \bar{f}) is a yes-instance of EXT RHF. ◀

As this is also a polynomial-time reduction, it implies following corollary.

► **Corollary 7.5.** EXT RHF is NP-complete.

We know that EXT RHF is polynomial-time solvable if the correspondence function is surjective. This leads to the question if $\kappa_1(\mathcal{I}) = |\{i \in I \mid \tau^{-1}(i) = \emptyset\}|$ is a good parameter for this problem for each instance $\mathcal{I} = (H, \tau, f)$ with $H = (V, (s_i)_{i \in I})$, $\tau : V \rightarrow I$, $f : V \rightarrow \{0, 1, 2\}$, somehow measuring the *distance from triviality* again. In other words, we try to use parameterized complexity to study the phenomenon that classical function properties as surjectivity seem to be crucial for finding polynomial-time algorithms for EXT RHF.

► **Theorem 7.6.** (*) κ_1 -EXT RHF is $W[3]$ -complete.

In the long version, we consider an explicit XP-algorithm for κ_1 -EXT RHF; membership in XP already follows from Theorem 7.6. We are discussing other parameterizations that either lead to para-NP-hardness results (Theorem 7.7) or to FPT-results (Theorem 7.9).

► **Theorem 7.7.** (*) κ_ζ -EXT RHF is para-NP-hard for each parameterization described by $\zeta \in \{|f^{-1}(0)|, |f^{-1}(1)|, |f^{-1}(2)|, |f^{-1}(\{0, 2\})|\}$.

We can use one of the reductions of Theorem 7.7 to prove transversal-hardness.

► **Theorem 7.8.** (*) If there would be an algorithm to enumerate all minimal Rhf of an instance (H, τ) with polynomial delay, then there is an algorithm that enumerates all minimal hitting sets of a hypergraph H' with polynomial delay.

► **Theorem 7.9.** κ_ζ -EXT RHF \in FPT for $\zeta \in \{\omega(2 - f), |f^{-1}(\{0, 1\})|\}$.

Proof (Sketch). Let $(H = (X, \hat{S}), \tau, f)$ be an instance. The idea is to walk through all functions $g : X \rightarrow \{0, 1, 2\}$ with $f \leq g$ and test if g is a minimal Rhf. This runs in FPT-time. Furthermore, we can check in polynomial time if a function is a minimal Rhf (by modifying Algorithm 1). Hence, there are $2^{\omega(2-f(x))}$ or $3^{|f^{-1}(\{0,1\})|}$ many possibilities for g . ◀

8 Conclusions

We have generalized the notion of Roman domination towards hypergraphs by introducing the definition of Rhf. We have proven that all minimal Rhf can be enumerated with polynomial delay if the correspondence function is surjective. This can be seen as a technical answer to our question what causes Roman domination to behave different from classical domination with respect to polynomial-delay enumerability. When the correspondence is not surjective, RHF rather behaves like DS; in particular, its extension problem is W[3]-complete when parameterized by the given pre-solution's weight, and we observe that it is transversal-hard to enumerate all minimal Rhf.

The main open problems in the context of this paper are the following ones:

- How tight is transversal hardness linked to the NP-hardness of a related extension problem? In the line of the studies in this paper, these links were pretty tight. But in general, only one direction is clear: if extensibility can be decided in polynomial time, then enumeration is possible with polynomial delay. For an even more general discussion in this direction, cf. [12, 17, 32, 37]. Also, in [31] graph problems related to Roman domination were studied and there, both polynomial-delay enumeration was shown and NP-hardness of the corresponding extension problem.
- We also do not know if the polynomial-delay enumerability questions that we discussed are really equivalent to the polynomial-delay enumerability of minimal hitting sets.
- We mentioned in the introduction that RD is in FPT, when parameterized in a dual way, meaning, in this case, by $n - k$, where n is the number of vertices of the graph and k is an upper-bound on the weight of the Rdf. It might be interesting to have similar results for the two generalizations of Roman domination introduced in this paper. However, now it is not very clear what the “dual” of the $\omega(f)$ -parameterization should be.

We are currently looking for non-trivial graph-classes where bounded-EXT RD is solvable in polynomial time, hence looking onto another tractability frontier.


Notice that up to quite recently, only a handful of (natural) problems have been known to be complete for W[3]. Even today, apart from the extension problems and their relatives that we mentioned throughout this paper, we only know of the problems shown in [11, 14, 15]. Seeing more and more problems these days that are complete for W[3] adds new interest to W[3]. It might be time to attack the 25-years-old open question if $W[3] = W^*[3]$, see [18]. According to [15], the current status is: $W[3] \subseteq W^*[3] \subseteq W[4]$.

References

- 1 A. Aazami, J. Cheriyan, and K. R. Jampani. Approximation algorithms and hardness results for packing element-disjoint Steiner trees in planar graphs. *Algorithmica*, 63(1-2):425–456, 2012. doi:10.1007/S00453-011-9540-3.
- 2 F. N. Abu-Khzam, C. Bazgan, M. Chopin, and H. Fernau. Data reductions and combinatorial bounds for improved approximation algorithms. *Journal of Computer and System Sciences*, 82(3):503–520, 2016. doi:10.1016/J.JCSS.2015.11.010.
- 3 F. N. Abu-Khzam, H. Fernau, and K. Mann. Roman census: Enumerating and counting Roman dominating functions on graph classes. In J. Leroux, S. Lombardy, and D. Peleg, editors, *48th International Symposium on Mathematical Foundations of Computer Science, MFCS*, volume 272 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.MFCS.2023.6.
- 4 F. N. Abu-Khzam, H. Fernau, and K. Mann. Minimal Roman dominating functions: Extensions and enumeration. *Algorithmica*, 86:1862–1887, 2024. doi:10.1007/s00453-024-01211-w.
- 5 B. D. Acharya. Domination in hypergraphs. *AKCE International Journal of Graphs and Combinatorics*, 4(2):117–126, 2007.
- 6 S. Bermudo and H. Fernau. Combinatorics for smaller kernels: The differential of a graph. *Theoretical Computer Science*, 562:330–345, 2015. doi:10.1016/J.TCS.2014.10.007.
- 7 S. Bermudo, H. Fernau, and J. M. Sigarreta. The differential and the Roman domination number of a graph. *Applicable Analysis and Discrete Mathematics*, 8:155–171, 2014.
- 8 T. Bläsius, T. Friedrich, J. Lischeid, K. Meeks, and M. Schirneck. Efficiently enumerating hitting sets of hypergraphs arising in data profiling. In *Algorithm Engineering and Experiments (ALENEX)*, pages 130–143. SIAM, 2019. doi:10.1137/1.9781611975499.11.
- 9 T. Bläsius, T. Friedrich, J. Lischeid, K. Meeks, and M. Schirneck. Efficiently enumerating hitting sets of hypergraphs arising in data profiling. *Journal of Computer and System Sciences*, 124:192–213, 2022. doi:10.1016/J.JCSS.2021.10.002.
- 10 T. Bläsius, T. Friedrich, and M. Schirneck. The complexity of dependency detection and discovery in relational databases. *Theoretical Computer Science*, 900:79–96, 2022. doi:10.1016/J.TCS.2021.11.020.
- 11 H. L. Bodlaender, C. Groenland, and M. Pilipczuk. On the complexity of problems on tree-structured graphs. Technical Report arXiv:2208.12543v3, ArXiv, Cornell University, 2022. doi:10.48550/arXiv:2208.12543v3.
- 12 F. Capelli and Y. Strozecki. Incremental delay enumeration: Space and time. *Discrete Applied Mathematics*, 268:179–190, 2019. doi:10.1016/J.DAM.2018.06.038.
- 13 K. Casel, H. Fernau, M. Khosravian Ghadikolaei, J. Monnot, and F. Sikora. On the complexity of solution extension of optimization problems. *Theoretical Computer Science*, 904:48–65, 2022. doi:10.1016/j.tcs.2021.10.017.
- 14 J. Chen and F. Zhang. On product covering in 3-tier supply chain models: Natural complete problems for $W[3]$ and $W[4]$. *Theoretical Computer Science*, 363(3):278–288, 2006. doi:10.1016/J.TCS.2006.07.016.
- 15 Y. Chen, J. Flum, and M. Grohe. An analysis of the W^* -hierarchy. *The Journal of Symbolic Logic*, 72(2):513–534, 2007. doi:10.2178/JSL/1185803622.
- 16 E. J. Cockayne, P. A. Dreyer Jr., S. M. Hedetniemi, and S. T. Hedetniemi. Roman domination in graphs. *Discrete Mathematics*, 278:11–22, 2004. doi:10.1016/J.DISC.2003.06.004.
- 17 N. Creignou, M. Kröll, R. Pichler, S. Skritek, and H. Vollmer. A complexity theory for hard enumeration problems. *Discrete Applied Mathematics*, 268:191–209, 2019. doi:10.1016/J.DAM.2019.02.025.
- 18 R. G. Downey and M. R. Fellows. Threshold dominating sets and an improved characterization of $W[2]$. *Theoretical Computer Science*, 209(1-2):123–140, 1998. doi:10.1016/S0304-3975(97)00101-1.
- 19 R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

- 20 P. A. Dreyer. *Applications and Variations of Domination in Graphs*. PhD thesis, Rutgers University, New Jersey, USA, 2000.
- 21 T. Eiter and G. Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, 24(6):1278–1304, 1995. doi:10.1137/S0097539793250299.
- 22 H. Fernau. ROMAN DOMINATION: a parameterized perspective. *International Journal of Computer Mathematics*, 85:25–38, 2008. doi:10.1080/00207160701374376.
- 23 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.
- 24 A. Gainer-Dewar and P. Vera-Licona. The minimal hitting set generation problem: Algorithms and computation. *SIAM Journal of Discrete Mathematics*, 31(1):63–100, 2017. doi:10.1137/15M1055024.
- 25 G. Gogic, C. H. Papadimitriou, and M. Sideri. Incremental recompilation of knowledge. *Journal of Artificial Intelligence Research*, 8:23–37, 1998. doi:10.1613/JAIR.380.
- 26 J. Guo, F. Hüffner, and R. Niedermeier. A structural view on parameterizing problems: distance from triviality. In R. Downey, M. Fellows, and F. Dehne, editors, *International Workshop on Parameterized and Exact Computation IWPEC 2004*, volume 3162 of *LNCS*, pages 162–173. Springer, 2004. doi:10.1007/978-3-540-28639-4_15.
- 27 T. W. Haynes, S.T. Hedetniemi, and M. A. Henning, editors. *Topics in Domination in Graphs*, volume 64 of *Developments in Mathematics*. Springer, 2020.
- 28 M. M. Kanté, V. Limouzy, A. Mary, and L. Nourine. Enumeration of minimal dominating sets and variants. In O. Owe, M. Steffen, and J. A. Telle, editors, *Fundamentals of Computation Theory — 18th International Symposium, FCT*, volume 6914 of *LNCS*, pages 298–309. Springer, 2011. doi:10.1007/978-3-642-22953-4_26.
- 29 M. M. Kanté, V. Limouzy, A. Mary, and L. Nourine. On the enumeration of minimal dominating sets and related notions. *SIAM Journal of Discrete Mathematics*, 28(4):1916–1929, 2014. doi:10.1137/120862612.
- 30 M. Liedloff, T. Kloks, J. Liu, and S.-L. Peng. Efficient algorithms for Roman domination on some classes of graphs. *Discrete Applied Mathematics*, 156(18):3400–3415, 2008. doi:10.1016/J.DAM.2008.01.011.
- 31 K. Mann and H. Fernau. Perfect Roman domination: Aspects of enumeration and parameterization. In A. A. Rescigno and U. Vaccaro, editors, *Combinatorial Algorithms (Proceeding 35th International Workshop on Combinatorial Algorithms IWOCA)*, volume 14764 of *LNCS*, pages 354–368. Springer, 2024. doi:10.1007/978-3-031-63021-7_27.
- 32 A. Mary. *Énumération des dominants minimaux d’un graphe*. PhD thesis, LIMOS, Université Blaise Pascal, Clermont-Ferrand, France, November 2013.
- 33 A. Mary and Y. Strozecki. Efficient enumeration of solutions produced by closure operations. *Discrete Mathematics & Theoretical Computer Science*, 21(3), 2019. doi:10.23638/DMTCS-21-3-22.
- 34 C. Padamutham and V. S. R. Palagiri. Algorithmic aspects of Roman domination in graphs. *Journal of Applied Mathematics and Computing*, 64:89–102, 2020. doi:10.1007/S12190-020-01345-4.
- 35 A. Pagourtzis, P. Penna, K. Schlude, K. Steinhöfel, D. S. Taylor, and P. Widmayer. Server placements, Roman domination and other dominating set variants. In R. A. Baeza-Yates, U. Montanari, and N. Santoro, editors, *Foundations of Information Technology in the Era of Networking and Mobile Computing, IFIP 17th World Computer Congress – TC1 Stream / 2nd IFIP International Conference on Theoretical Computer Science IFIP TCS*, pages 280–291. Kluwer, 2002. Also available as Technical Report 365, ETH Zürich, Institute of Theoretical Computer Science, 10/2001. doi:10.1007/978-0-387-35608-2_24.
- 36 R. C. Read and R. E. Tarjan. Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks*, 5:237–252, 1975. doi:10.1002/NET.1975.5.3.237.
- 37 Y. Strozecki. Enumeration complexity. *EATCS Bulletin*, 129, 2019.

Matching (Multi)Cut: Algorithms, Complexity, and Enumeration

Guilherme C. M. Gomes ✉ 

Departamento de Ciência da Computação, Universidade Federal de Minas Gerais,
Belo Horizonte, Brazil
CNRS/LIRMM, Montpellier, France

Emanuel Juliano ✉ 

Departamento de Ciência da Computação, Universidade Federal de Minas Gerais,
Belo Horizonte, Brazil

Gabriel Martins ✉ 

Departamento de Ciência da Computação, Universidade Federal de Minas Gerais,
Belo Horizonte, Brazil

Vinicius F. dos Santos ✉ 

Departamento de Ciência da Computação, Universidade Federal de Minas Gerais,
Belo Horizonte, Brazil

Abstract

A matching cut of a graph is a partition of its vertex set in two such that no vertex has more than one neighbor across the cut. The Matching Cut problem asks if a graph has a matching cut. This problem, and its generalization d -cut, has drawn considerable attention of the algorithms and complexity community in the last decade, becoming a canonical example for parameterized enumeration algorithms and kernelization. In this paper, we introduce and study a generalization of Matching Cut, which we have named Matching Multicut: can we partition the vertex set of a graph in at least ℓ parts such that no vertex has more than one neighbor outside its part? We investigate this question in several settings. We start by showing that, contrary to Matching Cut, it is NP-hard on cubic graphs but that, when ℓ is a parameter, it admits a quasi-linear kernel. We also show an $\mathcal{O}(\ell^{\frac{3}{2}})$ time exact exponential algorithm for general graphs and a $2^{\mathcal{O}(t \log t)} n^{\mathcal{O}(1)}$ time algorithm for graphs of treewidth at most t . We then turn our attention to parameterized enumeration aspects of matching multicuts. First, we generalize the quadratic kernel of Golovach et. al for Enum Matching Cut parameterized by vertex cover, then use it to design a quadratic kernel for Enum Matching (Multi)cut parameterized by vertex-deletion distance to co-cluster. Our final contributions are on the vertex-deletion distance to cluster parameterization, where we show an FPT-delay algorithm for Enum Matching Multicut but that no polynomial kernel exists unless $\text{NP} \subseteq \text{coNP/poly}$; we highlight that we have no such lower bound for Enum Matching Cut and consider it our main open question.

2012 ACM Subject Classification Theory of computation \rightarrow Parameterized complexity and exact algorithms

Keywords and phrases Matching Cut, Matching Multicut, Enumeration, Parameterized Complexity, Exact exponential algorithms

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.25

Related Version Full Version: <https://arxiv.org/abs/2407.02898>

Funding *Guilherme C. M. Gomes*: Partially funded by the European Union, project PACKENUM, grant number 101109317. Views and opinions expressed are however those of the author only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

Vinicius F. dos Santos: Partially funded by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), grants 312069/2021-9, 406036/2021-7 and 404479/2023-5 by Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG), grant APQ-01707-21.



© Guilherme C. M. Gomes, Emanuel Juliano, Gabriel Martins, and Vinicius F. dos Santos;
licensed under Creative Commons License CC-BY 4.0

19th International Symposium on Parameterized and Exact Computation (IPEC 2024).

Editors: Édouard Bonnet and Paweł Rzażewski; Article No. 25; pp. 25:1–25:15

Leibniz International Proceedings in Informatics



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

1 Introduction

A matching M in a graph G is a subset of the edges of G such that no vertex is the endpoint of more than one edge in M . Matchings are one of the most fundamental concepts in graph theory, with whole books dedicated to them [33, 34]. A cut of a graph G is a partition of its vertex set into two non-empty sets and we say that the set of edges between them is an edge cut. A matching cut is a edge cut that is also a matching. Not all graphs admit a matching cut, and graphs admitting such kind of cuts were first considered by Graham [28], who called them decomposable graphs, to solve a problem in number theory. Other applications include fault-tolerant networks [20], multiplexing networks [1] and graph drawing [41]. The problem of recognizing graphs that do admit a matching cut, called MATCHING CUT, was studied by Chvátal [13], who proved that the problem is NP-complete even restricted to graphs of maximum degree four, while polynomial-time solvable in graphs of maximum degree three. The problem was reintroduced under the current terminology in [41] and, since then, it has been attracting much attention of the algorithms community. It has also been shown to remain NP-complete for several graph classes such as bipartite graphs of bounded degree [42], planar graphs of bounded degree or bounded girth [8] and P_t -free graphs (for large enough t) [21]. On the positive side, tractable cases include H -free graphs, i.e. graphs without an induced subgraph isomorphic to H , for some H , including P_6 , the path on 6 vertices [36]. For a more comprehensive overview and recent developments, we refer to [12, 38].

MATCHING CUT has also been studied from the parameterized perspective, with the minimum number of edges crossing the cut k being used as the natural parameter for this problem. The first parameterized algorithm for k was given by Marx et al. in [39]; they tackled the STABLE CUTSET problem using the treewidth reduction machinery and Courcelle's theorem, which yielded a very large dependency on k . We remark that MATCHING CUT on G is equivalent to finding a separator that is an independent set in the line graph of G . Using the compact tree decomposition framework of Cygan et al. [16], Aravind and Saxena [4] developed a $2^{\mathcal{O}(k \log k)} n^{\mathcal{O}(1)}$ time algorithm for MATCHING CUT. Komusiewicz et al. [31] presented a quadratic kernel for the vertex-deletion distance to cluster parameterization, as well as single exponential time FPT algorithms for this parameterization and for vertex-deletion distance to co-cluster; on the other hand, they gave a kernelization lower bound for the combined parameterization of treewidth plus the number of edges in the cut. Aravind et al. [3] presented FPT algorithms for neighborhood diversity, twin-cover and treewidth for MATCHING CUT; the latter had its running time improved by Gomes and Sau in [26].

One area in which matching cuts have drawn particular attention is in parameterized enumeration. Under this framework, our goal is to list all feasible solutions to a problem, e.g. all matching cuts of an input graph. Parameterized algorithms that do so are classified in two families: TotalFPT – where all solutions can be listed in FPT time – and DelayFPT – where the delay between outputting two solutions, i.e. the time between these outputs, is at most FPT. Based on the foundational work of Creignou et al. [14], Golovach et al. [25] defined the kernelization analogues of TotalFPT and DelayFPT. Also in [25], the authors developed several enumeration and kernelization algorithms for ENUM MATCHING CUT under the vertex cover, neighborhood diversity, modular width, and clique partition number parameterizations. They also studied the enumeration of minimal and maximal matching cuts in the form of the ENUM MINIMAL MC and ENUM MAXIMAL MC problems under some of the aforementioned parameterizations.

Similar problems to MATCHING CUT, as well as minimization and maximization questions [37], have also been considered. Their hardness follow directly from the problem definition. Another related problem, PERFECT MATCHING CUT, asks for the existence of a

perfect matching that is also a matching cut. Although its hardness does not follow directly from MATCHING CUT the problem is also NP-complete [30]. The recent survey by Le et al. [32] revisit and compare results on these variations. Some problems, however, can be seen as direct generalizations of MATCHING CUT. In the d -CUT problem, the goal is to partition the vertex set into two sets such that each vertex has at most d neighbors in the opposite set of the partition. Introduced in [26], d -CUT has been shown to be NP-complete for $(2d + 2)$ -regular graphs and it has been shown to admit FPT algorithms under several parameters such as the maximum number of edges crossing the cut [4], treewidth, vertex-deletion distance to cluster, and vertex-deletion distance to co-cluster. When $d = 1$ the problem is exactly MATCHING CUT. However, many cases that are tractable for $d = 1$ have been shown to become hard for d -CUT [35]. The other related problem arises in the context of graph convexity. To our purposes, a convexity is a family \mathcal{C} of subsets of a finite ground set X such that $X, \emptyset \in \mathcal{C}$ and \mathcal{C} is closed for intersection. Many graph convexities have been considered in the literature [2, 11, 17, 29], most of them motivated by families of paths. In this context, a subset S of vertices is convex if all paths of a given type between vertices of S contain only vertices of S . The most well-studied paths in the literature are shortest paths, induced paths and P_3 , the paths on three vertices. One of the problems studied in the graph convexity setting is the partition of the vertex set of a graph into convex sets. Note that, in the P_3 -convexity, this is equivalent to partition vertices in such a way that two vertices in a set S have no common neighbor outside S . Hence, partitioning into two P_3 -convex sets is equivalent to MATCHING CUT. The more general case has also been considered in [10, 27].

Our contributions. In this work we introduce the MATCHING MULTICUT problem, a novel generalization of MATCHING CUT. A *matching multicut on ℓ parts* of a graph G is a partition of its vertex set in $\{A_1, \dots, A_\ell\}$ such that each vertex in A_i has at most one neighbor outside of A_i . Note that this is quite different from a partition into P_3 -convex sets; in the latter, a vertex $v \in A_i$ may have one neighbor in *each* other A_j , while in the former, v may have one neighbor in $\bigcup_{j \neq i} A_j$. Formally, we study the following problem:

MATCHING MULTICUT

Instance: A graph G and an integer ℓ .

Question: Does G admit a matching multicut on at least ℓ parts?

We explore the complexity landscape of MATCHING MULTICUT under several settings that were previously considered for MATCHING CUT. Since the case $\ell = 2$ is exactly MATCHING CUT, the problem is trivially NP-hard. It is also trivially paraNP-hard for the natural parameter ℓ . We study its complexity for cubic graphs, exact exponential algorithms, structural parameterizations as well parameterized enumeration questions.

Contrary to the classic result of Chvátal showing the polynomial-time solvability of MATCHING CUT [13] for cubic graphs, we show that MATCHING MULTICUT is NP-hard even restricted to those graphs. On the other hand, the problem becomes fixed parameter tractable when parameterized by ℓ . Indeed, we show that the problem admits a quasi-linear kernel under this parameterization for subcubic graphs. We also show that the problem is FPT parameterized by treewidth. From the definition of the problem, there is a trivial $\ell^n n^{\mathcal{O}(1)}$ time algorithm for MATCHING MULTICUT by just enumerating all possible (ordered) partitions of $V(G)$. We improve this by showing that the problem can be solved in $\alpha_\ell^n n^{\mathcal{O}(1)}$ time, with $\alpha_\ell \leq \sqrt{\ell}$ for a graph on n vertices. Finally, we turn our attention to the enumeration of matching multicuts in the form of the ENUM MATCHING MULTICUT problem.

ENUM MATCHING MULTICUT

Instance: A graph G and an integer ℓ .

Enumerate: All matching multicuts of G on at least ℓ parts

Our first results in this direction are a polynomial-delay enumeration (PDE) kernel under vertex cover and a PDE kernel under vertex-deletion distance to co-cluster. Afterwards, we present a DelayFPT algorithm for enumerating matching multicuts of a graph parameterized by the vertex-deletion distance to cluster. For our final result, we show that, although ENUM MATCHING MULTICUT is in DelayFPT for the vertex-deletion distance to cluster parameter, MATCHING MULTICUT does not admit a polynomial kernel under the joint parameterization of distance to cluster, maximum cluster size and the number of parts of the cut, unless $\text{NP} \subseteq \text{coNP}/\text{poly}$. To prove this result, we show that SET PACKING has no polynomial kernel parameterized by the size of the ground set, which could be of independent interest. To the best of our knowledge, although expected, this result has not been explicitly stated before.

1.1 Preliminaries

We denote $\{1, 2, \dots, n\}$ by $[n]$. We say that a monotonically non-decreasing function f is *quasi-linear* if $f(n) \in \mathcal{O}(n \log^c n)$ for some constant c . We use standard graph-theoretic notation, and we consider simple undirected graphs without loops or multiple edges; see [6] for any undefined terminology. When the graph is clear from the context, the degree (that is, the number of neighbors) of a vertex v is denoted by $\deg(v)$, and the number of neighbors of a vertex v in a set $A \subseteq V(G)$ and its neighborhood in it are denoted by $\deg_A(v)$ and $N_A(v)$; we also define $N(S) = \bigcup_{v \in S} N(v) \setminus S$. The minimum degree and the maximum degree of a graph G are denoted by $\delta(G)$ and $\Delta(G)$, respectively. We say that G is cubic if $\deg(v) = 3$ for all $v \in V(G)$ and that G is subcubic if $\deg(v) \leq 3$. A *matching* M of G is a subset of edges of G such that no vertex of G is incident to more than one edge in M ; for simplicity, we define $V(M) = \bigcup_{uv \in M} \{u, v\}$ and refer to it as the set of *M -saturated vertices*. The *subgraph of G induced by X* is defined as $G[X] = (X, \{uv \in E(G) \mid u, v \in X\})$. The vertex-deletion distance to \mathcal{G} is the size of a minimum cardinality set $U \subseteq V(G)$ such that $G \setminus U = G[V(G) \setminus U]$ belongs to class \mathcal{G} ; in this case, U is called the \mathcal{G} -modulator. A graph G is a *cluster graph* if each connected component is a clique; G is a *co-cluster graph* if its complement is a cluster graph. A *vertex cover* of G is a set of vertices incident to every edge of G . A tree decomposition $(T, \{X_t\}_{t \in V(T)})$ of a connected graph G is such that T is a tree, $X_t \subseteq V(G)$ for all t and: (i) for every $uv \in E(G)$ there is some $t \in T$ where $u, v \in X_t$ and (ii) the nodes of T that contain $v \in V(G)$ form a subtree of T , for every v . The sets X_t are called the *bags* of the decomposition, the *width* of the decomposition is $\max_{t \in V(T)} |X_t| - 1$. The *treewidth* of G is the size of a tree decomposition of G of minimum width. For more on treewidth and, in particular, nice tree decompositions, we refer the reader to [15].

We refer the reader to [15, 18] for basic background on parameterized complexity, and we recall here only some basic definitions. A *parameterized problem* is a tuple (L, κ) where $L \subseteq \Sigma^*$ is a language and $\kappa : \Sigma^* \mapsto \mathbb{N}$ is a parameterization. For an instance $I = (x, k) \in \Sigma^* \times \mathbb{N}$, k is called the *parameter*. A parameterized problem is *fixed-parameter tractable* FPT if there exists an algorithm \mathcal{A} , a computable function f , and a constant c such that given an instance $I = (x, k)$, \mathcal{A} (called an *FPT algorithm*) correctly decides whether $I \in L$ in time bounded by $f(k) \cdot |I|^c$. A fundamental concept in parameterized complexity is that of *kernelization*; see [23] for a recent book on the topic. A kernelization algorithm, or just *kernel*, for a parameterized problem Π takes an instance (x, k) of the problem and, in time polynomial in $|x| + k$, outputs an instance (x', k') such that $|x'|, k' \leq g(k)$ for some function g , and

$(x, k) \in \Pi$ if and only if $(x', k') \in \Pi$. The function g is called the *size* of the kernel. A kernel is called *polynomial* (resp. *quadratic*, *linear*) if the function $g(k)$ is a polynomial (resp. quadratic, linear) function in k .

In terms of parameterized enumeration, we refer the reader to [14, 25] for a more comprehensive overview than what we give below. A *parameterized enumeration problem* is a triple (L, Sol, κ) where $L \subseteq \Sigma^*$ is a language, $\text{Sol} : \Sigma^+ \mapsto 2^{\Sigma^*}$ is the set of all viable solutions and $\kappa : \Sigma^* \mapsto \mathbb{N}$ is a parameterization. An instance to a parameterized enumeration problem is a pair (x, k) where $k = \kappa(x)$ and the goal is to produce $\text{Sol}(x)$. We say that an algorithm \mathcal{A} that takes (x, k) as input is a **TotalFPT** algorithm if it outputs $\text{Sol}(x)$ in FPT time. Naturally, several problems won't have $\text{Sol}(x)$ of FPT size. In this case, the best we can hope for is that the *delay* to outputting a new solution is FPT. If not only this is the case but also: (i) the time to the first solution, and (ii) the time from the final solution to the halting of the algorithm are also in FPT, then we say that the algorithm is a **DelayFPT** algorithm. Very recently, Golovach et al. [25] gave kernelization analogues to **TotalFPT** and **DelayFPT**, which they called *fully-polynomial enumeration kernel* (FPE) and *polynomial-delay enumeration kernel* (PDE), respectively. Formally, an FPE kernel is a pair of algorithms $\mathcal{A}, \mathcal{A}'$ called the *compressor*¹ and *lifting* algorithms, respectively, where:

- Given (x, k) , \mathcal{A} outputs (x', k') with $|x'|, k' \leq g(k)$ in time $\text{poly}(|x| + k)$ for some computable g .
- For each $s \in \text{Sol}(x')$, \mathcal{A}' computes a set S_s in time $\text{poly}(|x| + |x'| + k + k')$ such that $\{S_s \mid s \in \text{Sol}(x')\}$ is a partition of $\text{Sol}(x)$.

For PDE kernels, we replace the polynomial (total) time condition of \mathcal{A}' with *polynomial delay* on $|x| + |x'| + k + k'$.

2 (Sub)Cubic graphs

A result of Chvátal [13] from the 1980s shows that **MATCHING CUT** is polynomial-time solvable for subcubic graphs. Later, Moshi [40] showed that every connected subcubic graph on at least eight vertices has a matching cut. When dealing with **MATCHING MULTICUT**, the situation is not as simple. We first show that, if the number of components ℓ is part of the input, then **MATCHING MULTICUT** is NP-hard. However, we are able to prove a Moshi-like result, and show that, if ℓ is a parameter, then the problem admits a quasi-linear kernel.

2.1 NP-hardness

First, let us show a lemma and some helpful definitions for our construction.

► **Definition 1.** A graph G is *indivisible* if and only if G has no matching cut. A set of vertices $X \subseteq V(G)$ is said to be *indivisible* if the subgraph of G induced by X is indivisible.

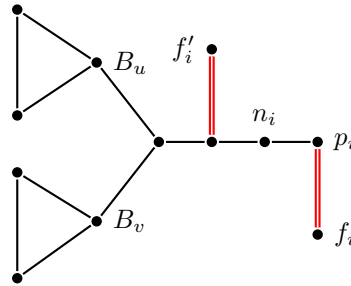
We remark that the above definition is a conservative notion of togetherness; i.e. we do not require that X is together in every matching cut of G , we require it to be together *regardless* of the remainder of the graph that contains it.

► **Definition 2.** Let $X \subset V(G)$ induce a connected subgraph of G with exactly one $u \in X$ such that $N(u) \setminus X \neq \emptyset$. If $|N(u) \setminus X| = 1$ we say $G[X]$ is a *pendant subgraph* of G and that X induces a *pendant subgraph* of G .

¹ This was named the *kernelization* algorithm in [25], but we reserve this term to the pair $\mathcal{A}, \mathcal{A}'$ itself.

► **Lemma 3.** Let $I = \{H_1, \dots, H_k\}$ be a set of maximal indivisible pendant subgraphs of G . Let v_1, \dots, v_k be pairwise distinct vertices so that $N(H_i) = \{v_i\}$. If (G, ℓ) is a yes-instance for some $\ell > k$, then there exists a matching multicut $\mathcal{P} = \{P_1, \dots, P_\ell\}$ with $P_i = V(H_i)$ for $1 \leq i \leq k$.

Construction. To construct our instance (H, ℓ) of MATCHING MULTICUT, we will reduce from an instance (G, k) of INDEPENDENT SET on cubic graphs, which is a well known NP-complete problem [24]. First, for each $u \in V(G)$, create a K_3 in H , label it as B_u , and let $B = \bigcup_{u \in V(G)} B_u$. Suppose that $E(G)$ has been arbitrarily ordered as $\{e_1, \dots, e_m\}$. For each edge in order, we add the gadget in Figure 1. The vertices in B_u and B_v are connected in such a way no vertex has more than three neighbors.

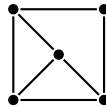


■ **Figure 1** Edge gadget for edge $e_i = uv$. Thick edges are assumed to be in any solution to MATCHING MULTICUT.

To connect our edge edges, we add an edge between n_i and p_{i+1} for every $i \in [m - 1]$ and between n_m and p_1 . With this the subgraph of H induced by the n_i 's and p_i 's is a cycle on $2m$ vertices. Finally, we set $\ell = 2m + k + 1$. Intuitively, the pendant vertices force that the entire cycle and the vertices between the triangles are contained in a single part of the multicut; as such, picking a triangle invalidates picking any other triangle at distance two.

► **Lemma 4.** $(H, 2|E(G)| + k + 1)$ is a YES-instance of MATCHING MULTICUT if and only if G has an independent set of size at least k . Moreover, MATCHING MULTICUT is NP-complete in subcubic graphs.

We can show in a very similar manner that MATCHING MULTICUT is NP-hard for cubic graphs. To do this, we replace the pendant vertices of H with the indivisible graph in Figure 2. The remainder of the argument follows as in the proof of the previous theorem.



■ **Figure 2** Indivisible pendant subgraph.

► **Corollary 5.** MATCHING MULTICUT is NP-complete in cubic graphs.

2.2 Quasi-linear kernel

We now present a quasi-linear kernel for the MATCHING MULTICUT problem in the case where the number of partitions ℓ is a parameter. In order to construct the kernel, we extend Moshi's result [40] and show the following theorem.

► **Theorem 6.** *Let G be a connected graph with $\Delta(G) \leq 3$. If $|V(G)| = \Omega(\ell \log^2 \ell)$, then G has a matching multicut that partitions the graph into ℓ parts.*

The key for proving Theorem 6 is to find a sufficiently large collection of vertex-disjoint cycles and construct a partition using some of them. In order to find these cycles, we first need to deal with vertices of degree at most 2.

► **Lemma 7.** *Let G be a connected subcubic graph, and let V_1 denote the vertices of G with degree 1. If $|V_1| \geq 3\ell$, then G has a matching multicut that partitions the graph into ℓ parts.*

Proof. Construct a matching M by greedily choosing edges that contain a vertex from V_1 . Because $\Delta(G) \leq 3$, $|M| \geq \frac{1}{3}|V_1| \geq \ell$. Moreover, $G - M$ contains at least $|M|$ components with an isolated vertex. ◀

► **Lemma 8.** *Let G be a connected subcubic graph with $|V(G)| = \Omega(\ell)$, and let V_2 denote the vertices of G with degree 2. If $|V_2| \geq \frac{9}{10}|V(G)|$, then G has a matching multicut that partitions the graph into ℓ parts.*

Assume that G is a subcubic graph that does not satisfy the degree conditions of Lemmas 7 and 8. That is, G does not have many degree-1 vertices nor a large proportion of degree-2 vertices. Our strategy to find a matching multicut for G involves using disjoint cycles. The intuition here is that if a cycle C is entirely contained within a part of a partition of G , each vertex $v \in C$ will have at most one edge crossing the partition. Therefore, cycles are a good starting point for partitioning the matching multicut. However, first we need to find disjoint cycles. For this purpose, we utilize a theorem due to Simonovits [43], which is a precise version of the well-known Erdős–Pósa theorem [19] in the context of subcubic graphs.

► **Theorem 9 (Simonovits '67).** *Let G be a connected graph with $\delta(G) \geq 2$. Let $V_{\geq 3}$ be the set of vertices of G with degree at least 3. Then, G has at least $|V_{\geq 3}|/(4 \log |V_{\geq 3}|)$ vertex disjoint cycles.*

It is worth mentioning that there exists an algorithmic approximation of Theorem 9 due to Brandstädt and Voss [9]. Therefore, all the theorems presented in this subsection are constructive and can be used to find a matching multicut of a subcubic graph.

Before moving to the main theorem of this subsection, we make some observations about the neighborhood of subcubic graphs. Let $N(v)$ be the set of vertices of G adjacent to v , and let $N[v] = N(v) \cup \{v\}$. More generally, let $N(S)$ be the set of vertices of $G - S$ that are adjacent to some vertex in S , and let $N[S] = N(S) \cup S$. We call $N(S)$ the open neighborhood and $N[S]$ the closed neighborhood. We denote by $N^2[S] := N[N[S]]$ the closed square neighborhood. Notice that for subcubic graphs, $|N^2[S]| \leq 10|S|$.

Proof of Theorem 6. Let G be a graph satisfying the conditions of Theorem 6. Let $V_1(G)$, $V_2(G)$, and $V_3(G)$ be the subsets of vertices of G with degrees 1, 2, and 3, respectively. If G satisfies the conditions of Lemmas 7 or 8, we are done. Now we can safely assume that $|V_1(G)| < 3\ell$ and $|V_2(G)| < \frac{9}{10}|V(G)|$.

Let G' be the graph obtained from G after recursively removing degree 1 vertices. Notice that each time a degree 1 vertex is removed, a vertex moves from V_2 to V_1 or from V_3 to V_2 . In both cases, the difference $|V_3| - |V_1|$ remains invariant; therefore, $|V_3(G')| \geq |V_3(G)| - |V_1(G)|$. Notice that any matching multicut of G' is also a matching multicut of G . Assume again that G' does not satisfy Lemma 8; in particular, this implies that $|V_3(G)| \geq |V(G)|/10$.

Now, G' satisfies the conditions of Theorem 9. Let $\{C_1, \dots, C_k\}$ be a collection of $k = |V_3(G')|/(4 \log |V_3(G')|) = \Omega(\ell \log \ell)$ vertex-disjoint sets such that $G[C_i]$ is a cycle. By giving a lower bound for the value of k , we also give a lower bound for $|V(G)|$. Later in the proof, we will need k such that $k^2 \geq c\ell|V(G')|$, but notice that there always exists a constant $c' > c$ such that if $|V(G)| \geq c'\ell \log^2 \ell$, the lower bound on k^2 is satisfied.

For each set of vertices C_i , if there is $v \in V(G') \setminus C_i$ with $|N(v) \cap C_i| \geq 2$, add v to C_i , that is, $C_i := C_i \cup \{v\}$. Notice that with this process, every vertex inside C_i has at least two neighbors inside C_i , therefore, $E(C_i, V(G') \setminus C_i)$ forms a matching cut.

We construct the matching multicut greedily. Let $M := \emptyset$ be the initial matching multicut and let $S := \emptyset$ be a collection of marked vertices. Assume that the sets C_i are ordered by size with $|C_1| \leq \dots \leq |C_k|$. Let C_i be a set in the first half of this ordering with no vertex marked, i.e., $C_i \cap S = \emptyset$. Add the edges with exactly one endpoint in C_i to M and mark $N^2[C_i]$, that is, $M := M \cup E(C_i, V(G') \setminus C_i)$ and $S := S \cup N^2[C_i]$. If no such C_i exists in the first half of the ordering, stop the process. We claim that in the end, M is indeed a matching multicut.

► **Lemma 10.** *If M is a set of edges constructed as above, then M is a matching multicut that divides G' into at least ℓ parts.*

It is easy to see that M is indeed a matching. Assuming otherwise, then there is a vertex v with two edges from M containing v . By previous observations, v must not belong to any set C_i whose border was added to M , thus $v \in V(G') \setminus (C_1 \cup \dots \cup C_k)$. If $|N(v) \cap C_i| \geq 2$, v would already have been added to C_i , so this cannot be the case. Hence, there are distinct sets C_i and C_j chosen in the algorithm with $|N(v) \cap C_i|, |N(v) \cap C_j| \geq 1$. Assume that C_i was chosen before C_j . As v is adjacent to a vertex of C_i and a vertex of C_j , there is a vertex of C_j in $N^2[C_i]$, which means that this vertex should have been marked, implying that this situation also cannot happen. We conclude that there is no vertex v with two edges from M containing v .

Now, we just need to check that during the process at least ℓ sets C_i were chosen so that the edges $E(C_i, V(G') \setminus C_i)$ were added to M . If this does not occur, we have that the size of the marked vertices $|S|$ is bounded:

$$|S| \leq (\ell - 1) \max_{i \leq k/2} \{|N^2[C_i]|\} \leq 10(\ell - 1)|C_{k/2}| \leq 10(\ell - 1) \frac{|V(G)|}{k/2} < \frac{k}{2}$$

In the first inequality, we are assuming the worst case where we have always added the largest squared neighbourhood. The second inequality follows from our previous bound on the size of this squared closed neighbourhood. The third inequality holds because the average size of the $k/2$ largest sets C_i is $2|V(G)|/k$, and the set $C_{k/2}$ has a size below this average. The last inequality follows from our lower bound on k . This concludes the proof by showing that M indeed divides G' into at least ℓ components, so it is a matching multicut. ◀

Notice that in the proof, we choose $|V(G)|$ in order to establish a lower bound on k^2 . We do not explicitly specify the choice of the constant c' such that $|V(G)| \geq c'\ell \log^2 \ell$. However, through a simple computation, it can be shown that $c' = 10^6$ is sufficient. We have not attempted to minimize the constants, but we believe that the value of c' can be reduced.

It follows from Theorem 6 that if we want to ask for a matching multicut that divides an n -vertex subcubic graph into $\ell = \mathcal{O}(n/\log^2 n)$ parts, the answer is trivially yes. On the other hand, Theorem 4 provides a construction of a subcubic graph and shows that it is NP-hard to determine if this graph has a matching multicut that divides it into $\Theta(n)$ parts. We leave it as an open question if it is possible to improve the asymptotic bound given by Theorem 6.

3 Exact Exponential Algorithm

We now turn our attention to developing an exact exponential algorithm through a similar approach used in [31]. For more on this type of algorithm and its associated terminology, we refer the reader to [22]. Our algorithm consists of four stopping rules, seven reduction and nine branching rules. At every step of the algorithm we have the sets $\{A_1, \dots, A_\ell, F\}$ such that $\varphi = \{A_1, \dots, A_\ell\}$ (unless any stopping rule is applicable) is a matching ℓ -multicut of the vertices of $V(G) \setminus F$. We set the size of the instance as the size of the set F , that is, how many free vertices are not assigned to any part yet. For simplicity, we assume that $\delta(G) \geq 2$. The arguments we use work with slight modifications to graphs of minimum degree one, but they would unnecessarily complicate the description of the algorithm.

Intuitively, stopping rules are applicable whenever a bad decision has been made by the branching algorithm and we must prune that branch. Reduction rules, on the other hand, are useful for cleaning up an instance after a branching step has been performed. Finally, our branching rules attempt to reduce the size of F as much as possible for each possibility. We follow the configurations given by Figure 3, and always branch on vertex v_1 . The main culprit behind our complexity is rule B8, which gives us a branching vector of the form $\{1\} \times \{3\}^{\ell-1}$ and branching factor $\sqrt[3]{\ell} \leq \alpha_\ell \leq \sqrt{\ell}$ which, for $\ell \gg 2$, will be our worst factor.

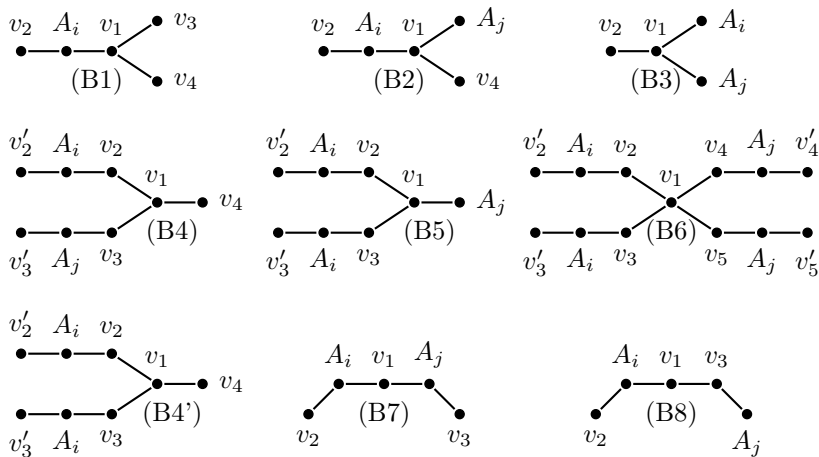


Figure 3 Branching configurations for MATCHING MULTICUT.

► **Theorem 11.** *MATCHING MULTICUT can be solved in $\alpha_\ell^n n^{\mathcal{O}(1)}$ time for a graph on n vertices, where $\alpha_\ell \leq \sqrt{\ell}$.*

4 FPT Algorithm by Treewidth

Let $(T, \{X_t\}_{t \in V(T)})$ be a nice tree decomposition of a graph G with n vertices, with T corresponding to the tree of the nice tree decomposition and X_t being the bag corresponding to vertex t . Suppose T is rooted at a vertex root , that $X_{\text{root}} = \emptyset$. Let V_t be the union of all the bags present in the subtree rooted at t . Finally, define $G_t = G[V_t]$.

Our goal is to have $c[t, \mathcal{P}, \text{Ext}] = \ell$ if and only if ℓ is the maximum integer such that (G_t, ℓ) is a YES instance of MATCHING MULTICUT that respects \mathcal{P} and Ext , which we now formally define. First, \mathcal{P} is a function $\mathcal{P} : X_t \mapsto X_t$ with $\mathcal{P}(v)$ corresponding to the vertex in X_t with the smallest label that is present in the same set as v in the partition. In other

words, \mathcal{P} is responsible for representing which partition of V_t we have assigned each vertex of X_t to. Note that $\mathcal{P}(v) \leq v$. Finally, $\text{Ext} : X_t \rightarrow \{0, 1\}$ is a function that signals whether each vertex in X_t has a neighbor in a different set in the partition. We denote by $\mathcal{P}|_{\bar{v}}$ the restriction of \mathcal{P} to $X_t \setminus \{v\}$. Note that we can easily update each value in $\mathcal{P}|_{\bar{v}}$ to account for the missing vertex: we pick the minimum element in $\mathcal{P}^{-1}(v) \setminus \{v\}$ and set it as the new root of the component previously identified by v .

► **Theorem 12.** *If given a nice tree decomposition of width k of the n -vertex graph G , there exists an algorithm that solves MATCHING MULTICUT in $2^{k \log k n}$ time.*

5 Matching Multicut Enumeration

5.1 Vertex Cover

In this section, we consider the parameterization of the matching multicut problem by the vertex cover number $\tau(G)$ of the input graph. This parameterization of ENUM MATCHING CUT was previously studied in [25]. We show that the enumeration kernel constructed by the authors of [25] is also an enumeration kernel for ENUM MATCHING MULTICUT. We assume that the vertex cover X of size $k \leq 2\tau(G)$ is given together with the input graph.

We describe the kernel constructed in [25]. Assume for simplicity that G contains no isolated vertices. Let $I = V(G) \setminus X$. Recall that I is an independent set. Denote by I_1 and $I_{\geq 2}$ the subsets of vertices of I with degree 1 and at least 2, respectively. We use the following marking procedure to label some vertices of I .

- (i) For every $x \in X$, mark an arbitrary vertex of $N(x) \cap I_1$ (if it exists).
- (ii) For every two distinct vertices $x, y \in X$, select an arbitrary set of $\min\{3, |N(x) \cap N(y) \cap I_{\geq 2}|\}$ vertices in $I_{\geq 2}$ adjacent to both x and y , and mark them for the pair $\{x, y\}$.

Denote by Z the set of marked vertices of I . Define $H = G[X \cup Z]$. Notice that $|V(H)| \leq 2|X| + 3\binom{|X|}{2} = \mathcal{O}(k^2)$. This completes the description of the basic compression algorithm that returns H . The key property of H is that it keeps all matching cuts of $G' = G - I_1$, including all matching multicuts of G' . Formally, we define $H' = H - I_1$. At this point, we can observe that the matching multicuts of H' and G' are in a one-to-one correspondence. With a few more technical details, we can prove Theorem 13.

► **Theorem 13.** *ENUM MATCHING MULTICUT admits a polynomial-delay enumeration kernel with $\mathcal{O}(k^2)$ vertices when parameterized by the vertex cover number k of the input graph.*

By Theorem 13, we have that matching multicuts can be listed with delay $k^{\mathcal{O}(k^2)} \cdot n^{\mathcal{O}(1)}$. We believe that this running time can be improved and the dependence on the vertex cover number can be made single exponential.

5.2 Distance to Co-cluster

A 3-approximation for this parameter can easily be computed in polynomial time: for every induced \overline{P}_3 , add all three of its vertices to the modulator. As such we assume that, along with (G, ℓ) , we are given a set S of size $k \leq 3\text{dcc}(G)$ so that $G \setminus S$ is a co-cluster graph. We break down our analysis in three cases: if $G \setminus S$ has at least three parts, two large parts, or neither of the previous two. For the first two, we essentially have that $G \setminus S$ is indivisible, while for the last one we may simply invoke the algorithm for the vertex cover parameterization.

► **Theorem 14.** *ENUM MATCHING MULTICUT admits a polynomial-delay enumeration kernel with $\mathcal{O}(k^2)$ vertices when parameterized by the distance to co-cluster k of the input graph.*

5.3 Distance to Cluster

In this section, we present a DelayFPT enumeration algorithm for ENUM MATCHING MULTICUT, parameterized by the vertex-deletion distance to cluster. We base our result on the quadratic kernel for MATCHING CUT given in [31]. The authors apply several reduction rules until they reach a kernel of size $\text{dc}(G)^{\mathcal{O}(1)}$. We use a subset of these rules as a starting point for our enumeration algorithm, then expand them a more careful analysis and needed technicalities for an enumeration algorithm. Formally, we prove the following theorem.

► **Theorem 15.** *There is an algorithm for ENUM MATCHING MULTICUT on n -vertex graphs with distance to cluster $\text{dc}(G) \leq t$ of delay $2^{\mathcal{O}(t^3 \log t)} + n^{\mathcal{O}(1)}$.*

Our strategy to enumerate all possible matching multicuts can be divided into 5 steps:

1. We apply reduction rules, similar to the kernel given in [31], spending $\text{poly}(|G|)$ time.
2. We enumerate all possible matching multicuts of a smaller instance of size $\mathcal{O}(t^3)$. This step takes a total time of $2^{\mathcal{O}(t^3 \log t)}$.
3. Given a matching multicut generated in step 2, we create an instance of ENUM SET PACKING, where the ground set has size t and the number of sets is potentially 2^t . All solutions are enumerated in total time $2^{\mathcal{O}(t^2)}$, and then each solution is extended to form a matching multicut.
4. Given a matching multicut from step 3, we increase the number of partitions by considering clusters of size 2 with only one edge to U . Now, we guarantee that we have at least ℓ partitions. As the number of matching multicuts with at least ℓ parts can be unbounded by ℓ , we worry about the delay of the enumeration and no longer with its total time.
5. We enumerate equivalent solutions for the original instance.

6 Kernelization lower bound for distance to cluster

Since we do have a DelayFPT algorithm for the vertex-deletion distance to cluster parameterization, it is natural to ask whether we can build a PDE kernel of polynomial size. In this section, we show this in the negative by presenting an exponential lower bound for MATCHING MULTICUT under this parameterization.

To obtain our result, we first show a kernelization lower bound for SET PACKING. In this problem, we are given a ground set X , a family $\mathcal{F} \subseteq 2^X$, and an integer k , and are asked to find $\mathcal{F}' \subseteq \mathcal{F}$ of size at least k such that for any $A, B \in \mathcal{F}'$ it holds that $A \cap B = \emptyset$. In particular, we prove Theorem 16.

► **Theorem 16.** *SET PACKING has no polynomial kernel when parameterized by $|X|$ unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.*

Our proof is based on an OR-cross-composition [5] from SET PACKING onto itself under the desired parameterization. To this end, we denote our input collection of SET PACKING instances by $\{(Y_1, \mathcal{E}_1, r_1), \dots, (Y_t, \mathcal{E}_t, r_t)\}$. Moreover, we can assume that $Y_i = \{y_1, \dots, y_n\}$ and $r_i = r$ for all $i \in [t]$ and, w.l.o.g, that $t = 2^\tau$ for some $\tau > 0$; the latter can be easily achieved by copying any one instance $2^\tau - t$ times and adding it to the input collection, which at most doubles this set if τ is the minimum integer such that $2^\tau \geq t$.

Construction. We construct our (X, \mathcal{F}, k) SET PACKING instance as follows. Our set X is partitioned into the set of input elements Y , index elements $S = \{s_0, s_1, \dots, s_r\}$, and a set of bits $\{b_{i,j}, \bar{b}_{i,j} \mid i \in [\tau], j \in [r]\}$. We define $\text{bits}_j(a)$ to be the set where $b_{i,j} \in \text{bits}_j(a)$ if and only if the i -th bit in the binary representation of a is 1, otherwise we have that $\bar{b}_{i,j} \in \text{bits}_j(a)$.

25:12 Matching (Multi)Cut: Algorithms, Complexity, and Enumeration

The family \mathcal{F} is partitioned in selector sets, identified as $\mathcal{T} = \{T_1, \dots, T_t\}$, and packing sets \mathcal{P} . Each T_a is defined as $T_a = \{s_0\} \cup \bigcup_{j \in r} \text{bits}_j(\bar{a})$, where \bar{a} is the (positive) bitwise complement of a , i.e. $a + \bar{a} = 2^r - 1$. As to our packing sets, for each input instance (Y, \mathcal{E}_a, r) , each $C_i \in \mathcal{E}_a$, and each $j \in [r]$, we add to \mathcal{P} the set $C_{a,i,j} = C_i \cup \text{bits}_j(a) \cup \{s_j\}$. Finally, we set $k = r + 1$. Intuitively, packing $T_a \in \mathcal{T}$ corresponds to solving instance (Y, \mathcal{E}_a, r) and, since every T_a has s_0 , only one of them can be picked. The way that our bits sets were distributed, picking T_a automatically excludes all elements in \mathcal{P} corresponding to sets present in another instance (Y, \mathcal{E}_c, r) . Finally, index elements S are used to ensure that at least one instance set is packed. The next observation follows immediately from the construction of our instance.

► **Observation 17.** *Instance (X, \mathcal{F}, k) is such that $|X| \leq |Y| + (r + 1)(1 + \log t)$ and $|C| \leq 1 + r \log t$ for all $C \in \mathcal{F}$.*

The proof of our main result, Theorem 18, follows from a simple polynomial parameter transformation from SET PACKING parameterized by the size of the ground set.

► **Theorem 18.** *When parameterized by the vertex-deletion distance to cluster, size of the maximum clique, and the number of parts of the cut, MATCHING MULTICUT does not admit a polynomial kernel unless $\text{NP} \subseteq \text{coNP/poly}$.*

7 Final Remarks

In this paper, we introduced and studied the MATCHING MULTICUT problem, a generalization of the well known MATCHING CUT problem, where we want to partition a graph G into at least ℓ parts so that no vertex has more than one neighbor outside of its own part. Specifically, we proved that the problem is NP-hard on subcubic graphs, but admits a quasi-linear kernel when parameterized by ℓ on this graph class. We also showed an $\ell^{\frac{3}{2}} n^{\mathcal{O}(1)}$ exact exponential algorithm based on branching for general graphs. In terms of parameterized complexity, aside from our aforementioned kernel, we give a $2^{\mathcal{O}(t \log t)} n^{\mathcal{O}(1)}$ time algorithm for graphs of treewidth at most t . Then, we move on to enumeration aspects, presenting polynomial-delay enumeration kernels for the vertex cover and distance to co-cluster parameterizations, the latter of which was an open problem for ENUM MATCHING CUT. Finally, we give a DelayFPT algorithm for the distance to cluster parameterization, and show that no polynomial-sized PDE kernel exists unless $\text{NP} \subseteq \text{coNP/poly}$. This last result is obtained by showing that SET PACKING has no polynomial kernel parameterized by the cardinality of the ground set.

For future work, we are interested in further exploring all aspects of this problem, such as graph classes and other structural parameterizations. As with MATCHING CUT, it seems interesting to study optimization and perfect variations of this problem, which may yield significant differences in complexity to MATCHING MULTICUT. While MAXIMUM MATCHING MULTICUT is NP-hard as PERFECT MATCHING CUT is NP-hard on 3-connected cubic planar bipartite graphs [7], the proof does not help in terms of W[1]-hardness. We believe that it in fact is W[1]-hard parameterized by $\ell +$ number of edges in the cut even on cubic graphs.

Our other questions of interest are mostly in the enumeration realm. In particular, we have no idea if it is possible to enumerate matching cuts on (sub)cubic graphs, and we consider it one of the main open problems in the matching cut literature. Finally, we are interested in understanding how to rule out the existence of TotalFPT and DelayFPT algorithms for a given problem and, ultimately, how to differentiate between problems that admit FPE and PDE kernels of polynomial size and those that do not.

References

- 1 Júlio Araújo, Nathann Cohen, Frédéric Giroire, and Frédéric Havet. Good edge-labelling of graphs. *Discrete Applied Mathematics*, 160(18):2502–2513, 2012. doi:10.1016/J.DAM.2011.07.021.
- 2 Rafael T Araújo, Rudini M Sampaio, Vinicius F dos Santos, and Jayme L Szwarcfiter. The convexity of induced paths of order three and applications: complexity aspects. *Discrete Applied Mathematics*, 237:33–42, 2018. doi:10.1016/J.DAM.2017.11.007.
- 3 N. R. Aravind, Subrahmanyam Kalyanasundaram, and Anjeneya Swami Kare. On structural parameterizations of the matching cut problem. In *Proc. of the 11th International Conference on Combinatorial Optimization and Applications (COCO A)*, volume 10628 of *LNCS*, pages 475–482, 2017. doi:10.1007/978-3-319-71147-8_34.
- 4 N. R. Aravind and Roopam Saxena. An FPT algorithm for matching cut and d -cut. In Paola Flocchini and Lucia Moura, editors, *Combinatorial Algorithms*, pages 531–543, Cham, 2021. Springer International Publishing. doi:10.1007/978-3-030-79987-8_37.
- 5 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Cross-composition: A new technique for kernelization lower bounds. In *Proc. of the 28th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 9 of *LIPICs*, pages 165–176, 2011. doi:10.4230/LIPICs.STACS.2011.165.
- 6 J.A. Bondy and U.S.R Murty. *Graph Theory*. Springer Publishing Company, Incorporated, 1st edition, 2008.
- 7 Édouard Bonnet, Dibyayan Chakraborty, and Julien Duron. Cutting barnette graphs perfectly is hard. In Daniël Paulusma and Bernard Ries, editors, *Graph-Theoretic Concepts in Computer Science*, pages 116–129, Cham, 2023. Springer Nature Switzerland. doi:10.1007/978-3-031-43380-1_9.
- 8 Paul Bonsma. The complexity of the matching-cut problem for planar graphs and other graph classes. *Journal of Graph Theory*, 62(2):109–126, 2009. doi:10.1002/JGT.20390.
- 9 Andreas Brandstädt and Heinz-Jürgen Voss. Short disjoint cycles in graphs with degree constraints. *Discrete Applied Mathematics*, 64(3):197–205, 1996. doi:10.1016/0166-218X(94)00133-X.
- 10 Carmen C. Centeno, Simone Dantas, Mitre Costa Dourado, Dieter Rautenbach, and Jayme Luiz Szwarcfiter. Convex partitions of graphs induced by paths of order three. *Discrete Mathematics & Theoretical Computer Science*, 12(Graph and Algorithms), 2010. doi:10.46298/DMTCS.502.
- 11 Carmen C. Centeno, Mitre C. Dourado, Lucia Draque Penso, Dieter Rautenbach, and Jayme L. Szwarcfiter. Irreversible conversion of graphs. *Theoretical Computer Science*, 412(29):3693–3700, 2011. doi:10.1016/J.TCS.2011.03.029.
- 12 Chi-Yeh Chen, Sun-Yuan Hsieh, Hoàng-Oanh Le, Van Bang Le, and Sheng-Lung Peng. Matching cut in graphs with large minimum degree. *Algorithmica*, 83:1238–1255, 2021. doi:10.1007/S00453-020-00782-8.
- 13 Vasek Chvátal. Recognizing decomposable graphs. *Journal of Graph Theory*, 8(1):51–53, 1984. doi:10.1002/JGT.3190080106.
- 14 Nadia Creignou, Arne Meier, Julian-Steffen Müller, Johannes Schmidt, and Heribert Vollmer. Paradigms for parameterized enumeration. *Theory of Computing Systems*, 60(4):737–758, May 2017. doi:10.1007/s00224-016-9702-4.
- 15 Marek Cygan, Fedor V Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 3. Springer, 2015. doi:10.1007/978-3-319-21275-3.
- 16 Marek Cygan, Paweł Komosa, Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, Saket Saurabh, and Magnus Wahlström. Randomized contractions meet lean decompositions. *ACM Trans. Algorithms*, 17(1), December 2021. doi:10.1145/3426738.
- 17 Mitre C Dourado, Fábio Protti, and Jayme L Szwarcfiter. Complexity results related to monophonic convexity. *Discrete Applied Mathematics*, 158(12):1268–1274, 2010. doi:10.1016/J.DAM.2009.11.016.

- 18 Rodney G Downey and Michael R Fellows. *Fundamentals of parameterized complexity*, volume 4. Springer, 2013. doi:10.1007/978-1-4471-5559-1.
- 19 P. Erdős and L. Pósa. On independent circuits contained in a graph. *Canadian Journal of Mathematics*, 17:347–352, 1965. doi:10.4153/CJM-1965-035-8.
- 20 Arthur M Farley and Andrzej Proskurowski. Networks immune to isolated line failures. *Networks*, 12(4):393–403, 1982. doi:10.1002/NET.3230120404.
- 21 Carl Feghali. A note on matching-cut in P_t -free graphs. *Information Processing Letters*, 179:106294, 2023. doi:10.1016/J.IPL.2022.106294.
- 22 Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Springer-Verlag, Berlin, Heidelberg, 1st edition, 2010.
- 23 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019.
- 24 M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified np-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976. doi:10.1016/0304-3975(76)90059-1.
- 25 Petr A. Golovach, Christian Komusiewicz, Dieter Kratsch, et al. Refined notions of parameterized enumeration kernels with applications to matching cut enumeration. *Journal of Computer and System Sciences*, 123:76–102, 2022. doi:10.1016/J.JCSS.2021.07.005.
- 26 Guilherme C. M. Gomes and Ignasi Sau. Finding cuts of bounded degree: Complexity, FPT and exact algorithms, and kernelization. *Algorithmica*, 83(6):1677–1706, June 2021. doi:10.1007/s00453-021-00798-8.
- 27 Lucía M. González, Luciano N. Grippo, Martín D. Safe, and Vinicius F. dos Santos. Covering graphs with convex sets and partitioning graphs into convex sets. *Information Processing Letters*, 158:105944, 2020. doi:10.1016/J.IPL.2020.105944.
- 28 Ron L. Graham. On primitive graphs and optimal vertex assignments. *Annals of the New York academy of sciences*, 175(1):170–186, 1970.
- 29 Frank Harary and Juhani Nieminen. Convexity in graphs. *Journal of Differential Geometry*, 16(2):185–190, 1981.
- 30 Pinar Heggernes and Jan Arne Telle. Partitioning graphs into generalized dominating sets. *Nord. J. Comput.*, 5(2):128–142, 1998.
- 31 Christian Komusiewicz, Dieter Kratsch, and Van Bang Le. Matching cut: Kernelization, single-exponential time FPT, and exact exponential algorithms. *Discrete Applied Mathematics*, 283:44–58, 2020. doi:10.1016/j.dam.2019.12.010.
- 32 Van Bang Le, Felicia Lucke, Daniël Paulusma, and Bernard Ries. Maximizing matching cuts. *arXiv preprint arXiv:2312.12960*, 2023. doi:10.48550/arXiv.2312.12960.
- 33 László Lovász and Michael D. Plummer. *Matching theory*, volume 367. American Mathematical Soc., 2009.
- 34 Cláudio L. Lucchesi and U.S.R. Murty. *Perfect Matchings: A Theory of Matching Covered Graphs*. Algorithms and Computation in Mathematics. Springer Nature Switzerland, Imprint: Springer, 2024.
- 35 Felicia Lucke, Ali Momeni, Daniël Paulusma, and Siani Smith. Finding d -cuts in graphs of bounded diameter, graphs of bounded radius and h -free graphs. *arXiv preprint arXiv:2404.11389*, 2024. doi:10.48550/arXiv.2404.11389.
- 36 Felicia Lucke, Daniël Paulusma, and Bernard Ries. On the complexity of matching cut for graphs of bounded radius and H -free graphs. *Theoretical Computer Science*, 936:33–42, 2022. doi:10.1016/J.TCS.2022.09.014.
- 37 Felicia Lucke, Daniël Paulusma, and Bernard Ries. Dichotomies for maximum matching cut: h -freeness, bounded diameter, bounded radius. *arXiv preprint arXiv:2304.01099*, 2023. doi:10.48550/arXiv.2304.01099.
- 38 Felicia Lucke, Daniël Paulusma, and Bernard Ries. Finding matching cuts in H -free graphs. *Algorithmica*, 85(10):3290–3322, 2023. doi:10.1007/S00453-023-01137-9.
- 39 Dániel Marx, Barry O’sullivan, and Igor Razgon. Finding small separators in linear time via treewidth reduction. *ACM Trans. Algorithms*, 9(4), October 2013. doi:10.1145/2500119.

- 40 Augustine M Moshi. Matching cutsets in graphs. *Journal of Graph Theory*, 13(5):527–536, 1989. doi:10.1002/JGT.3190130502.
- 41 Maurizio Patrignani and Maurizio Pizzonia. The complexity of the matching-cut problem. In *Graph-Theoretic Concepts in Computer Science: 27th International Workshop, WG 2001 Boltenhagen, Germany, June 14–16, 2001 Proceedings 27*, pages 284–295. Springer, 2001. doi:10.1007/3-540-45477-2_26.
- 42 Bert Randerath et al. On stable cutsets in line graphs. *Theoretical Computer Science*, 301(1-3):463–475, 2003. doi:10.1016/S0304-3975(03)00048-3.
- 43 Miklós Simonovits. A new proof and generalizations of a theorem of Erdős and Pósa on graphs without $k + 1$ independent circuits. *Acta Mathematica Hungarica*, 18(1-2):191–206, 1967.

The PACE 2024 Parameterized Algorithms and Computational Experiments Challenge: One-Sided Crossing Minimization

Philipp Kindermann ✉ 

Trier University, Germany

Fabian Klute ✉ 

Polytechnic University of Catalonia, Barcelona, Spain

Soeren Terziadis ✉ 

Eindhoven University of Technology, The Netherlands

Abstract

This article is a report by the challenge organizers on the 9th Parameterized Algorithms and Computational Experiments Challenge (PACE 2024). As was common in previous iterations of the competition, this year's iteration implemented an exact and heuristic track for a parameterized problem that has gained attention in the theory community. This year's challenge is about the ONE-SIDED CROSSING MINIMIZATION PROBLEM (OSCM). In the exact track, the competition participants were asked to develop an exact algorithm that can solve as many instances as possible from a benchmark set of 100 instances – with a time limit of 30 minutes per instance. In the heuristic track, the task must be accomplished within 5 minutes, however, the result in this track is not required to be optimal. New this year is the parameterized track, which has the same rules as the exact track, but instances are guaranteed to have small cutwidth. As in previous iterations, the organizers handed out awards to the best solutions in all tracks and to the best student submissions.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Graph algorithms analysis; Mathematics of computing → Graph algorithms

Keywords and phrases One-Sided Crossing Minimization, Algorithm Engineering, FPT, Heuristics

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.26

Acknowledgements The prize money (€4000) was generously provided by Networks [33], an NWO Gravitation project of the University of Amsterdam, Eindhoven University of Technology, Leiden University and the Center for Mathematics and Computer Science (CWI). We are grateful to the whole optil.io team, especially to Jan Badura for the fruitful collaboration and for hosting the competition at the optil.io online judge system. We also thank Markus Wallinger, who made his exact solver available to the organizers prior to the competition for internal evaluations [38].

1 Introduction: History and Timeline of PACE

The Parameterized Algorithms and Computational Experiments Challenge (PACE) was conceived in Fall 2015 to deepen the relationship between parameterized algorithms and practice. The declared mission of the PACE challenge is to

- bridge the divide between the theory of algorithm design and analysis, and the practice of algorithm engineering,
- inspire new theoretical developments,
- investigate in how far theoretical algorithms from parameterized complexity and related fields are competitive in practice,
- produce universally accessible libraries of implementations and repositories of benchmark instances,
- encourage the dissemination of these findings in scientific papers.



© Philipp Kindermann, Fabian Klute, and Soeren Terziadis;
licensed under Creative Commons License CC-BY 4.0

19th International Symposium on Parameterized and Exact Computation (IPEC 2024).

Editors: Édouard Bonnet and Paweł Rzażewski; Article No. 26; pp. 26:1–26:20

Leibniz International Proceedings in Informatics



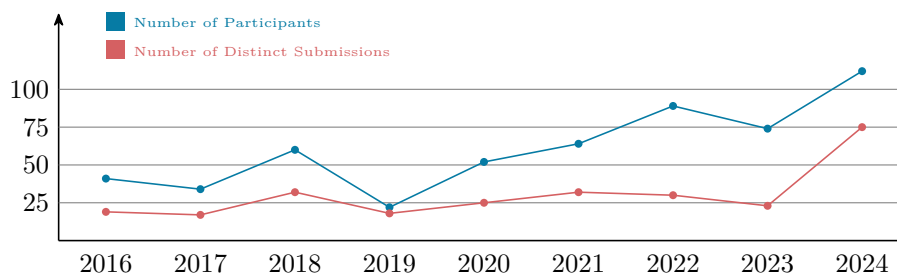
LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

26:2 PACE 2024: One-Sided Crossing Minimization

Previous iterations of the PACE challenge addressed a variety of graph optimization problems. Specifically, the previous iterations considered

PACE 2016	TREewidth and UNDIRECTED-FEEDBACK-VERTEX-SET	[6];
PACE 2017	TREewidth and MINIMUM FILL-IN	[7];
PACE 2018	STEINER TREE	[5];
PACE 2019	VERTEX-COVER and HYPERTREewidth	[13];
PACE 2020	TREEDePTH	[28];
PACE 2021	CLUSTER-EDITING	[25];
PACE 2022	DIRECTED-FEEDBACK-VERTEX-SET	[18];
PACE 2023	TWINWIDTH	[3].

Several of the previous iterations also contained more specialized tracks. Starting with the first iteration of PACE, many participants from all over the world were interested in the challenge and quickly established PACE as a highly competitive challenge. The competition attracted a record number of 112 participants this year, resulting in 75 submissions; see Figure 1.

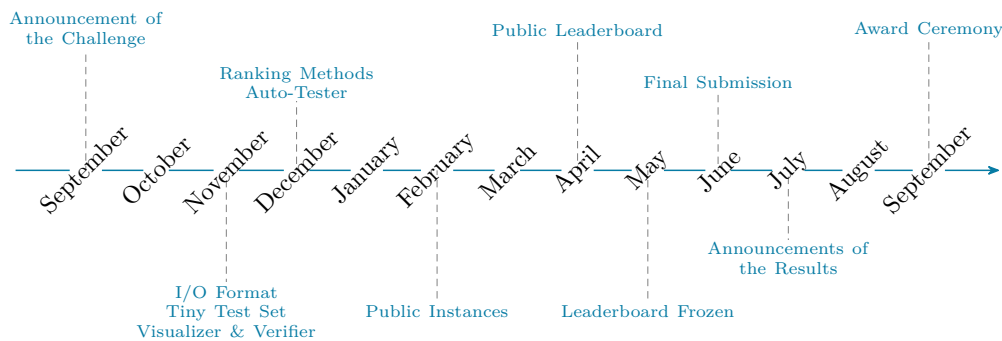


■ **Figure 1** Overview of the number of participants and distinct submissions of the PACE challenge over the years. Teams submitting multiple times and to multiple tracks are counted multiple times.

Papers inspired by concrete implementations created in the context of the PACE challenge were published in prestigious conferences such as ACDA, ALENEX, ESA (Track B), SEA, and WADS. The instances provided by PACE have also often been used to showcase further algorithmic improvements by being used as an established benchmark, ranging also to other competitions such as the famous SAT competition [2].

This report contains the relevant information on the ninth PACE challenge. The problem chosen was ONE-SIDED CROSSING MINIMIZATION, an important classical graph drawing problem with applications in hierarchical graph drawing, e.g., in the Sugiyama framework [37]. The challenge featured three tracks: an exact track, a heuristic track and a parameterized track. In the exact track, the task was to find an optimal solution of a given instance within 30 minutes and a memory limit of 8 GB. The instances in the exact track were guaranteed to have a low number of crossings relative to the instance size. In the heuristic track the number of crossings could be very large and the task was to compute a valid (but not necessarily optimal) solution with as few crossings as possible within a time limit of 5 minutes and a memory limit of 8 GB. The parameterized track used the same rules as the exact track (compute an optimal solution within 30 minutes using 8GB). However here the number of crossings was not guaranteed to be small and instead the cutwidth of the given instances was low. Additionally a witness for this low cutwidth was provided for every instance.

The timeline of the challenge started with the announcement in September 2023. Details about the input and output format were provided in November 2023 together with a tiny test set to allow the participants to start with the challenge. In addition, a small visualizer was provided, which can be used to view instances and solutions. Alongside we provided (via pip and source download) a verifier, which could be used to verify a given solution using a set of crossing counting algorithms. The concrete ranking methods for both tracks were published in December 2023. At the same time we provided a JUnit-like auto-tester, which runs a solver against a given set of instances and compares the output to provided (optimal) solutions. In early February 2024 the public instances and details about the benchmark set were published. Additionally a Github repository was made available with the intention that participants could add their own created test instances to make them available to all participating teams. The public leaderboard on the optil.io platform was opened in April 2024 and frozen on May 20th 2024. This allowed the participants to test their solvers on the public instances and provided a provisional ranking. The final version of the submission for the solver code was due on the ninth of June 2024 and the descriptions of the solvers had to be submitted until June 23. Afterwards, the submissions were evaluated on the private instances, which were similar in structure to the public instances but unknown to the participants. The results of this evaluation were announced in July 2024 (a correction of the ranking was issued on the ninth of August 2024), and the award ceremony took place during the International Symposium on Parameterized and Exact Computation (IPEC) 2024 at Royal Holloway University of London in Egham. The complete timeline can be found in Figure 2.



■ **Figure 2** Timeline of the PACE challenge in 2024 (the diagram ranges from September 2023 to September 2024). The next iteration of the PACE challenge for 2025 was announced during the award ceremony.

2 The Challenge Problem: One-Sided Crossing Minimization

This year's challenge was about the problem ONE-SIDED CROSSING MINIMIZATION (OSCM). This problem involves arranging the nodes of a bipartite graph on two layers (typically horizontal), with one of the layers fixed, aiming to minimize the number of edge crossings. More formally:

ONE-SIDED CROSSING MINIMIZATION (OSCM)

Input: A bipartite graph $G = ((A \cup B), E)$, and a linear order of A .

Output: A linear order of B .

Measure: The number of edge crossings in a straight-line drawing of G with A and B on two parallel lines, following their linear order.

OSCM is one of the basic building blocks used for drawing hierarchical graphs [37]. While easy to state it turns out that the problem is difficult to solve efficiently. As many graph drawing problems OSCM is well known to be NP-hard [14], even for star forests of degree 4 [32] and for trees [9]. Obtaining exact solutions in practice is commonly done using SAT or ILP formulations of the problem [23].

Turning to other exact algorithms, the problem can be solved in FPT time using the number of crossings as the parameter. Dujmovic and Whitesides [11] gave the first algorithm in 2004. Subsequent research [12, 26] pushed the running time down to $O(k2^{\sqrt{2k}} + n)$, where the exponent $\sqrt{2k}$ is asymptotically optimal assuming the exponential time hypothesis.

Aside from exact solutions, OSCM does admit a constant-factor approximation [15]. More importantly for practical considerations though, OSCM admits good heuristics. Two of the best-known ones are the *barycenter* and the *median heuristic*. As the names suggest, in the former each vertex is placed in the barycenter of its neighbors and in the latter in the median position of its neighbors. These very simple heuristics even come with some theoretical guarantees. For example, the barycenter heuristic yields a $O(\sqrt{n})$ approximation [31].

For an extended overview, see Chapter 13.5 of the Handbook of Graph Drawing [22].

In the parameterized track, all instances have small cutwidth. Given a graph $G = (V, E)$ and a linear ordering π on V , the *cutwidth* of (G, π) is the maximum number of edges that cross any partitioning of V into earlier and later subsets of π , that is, $\max_{1 \leq i < |V|} |\{(u, v) \in E \mid \pi(u) \leq i < \pi(v)\}|$. The *cutwidth* of G is the minimum cutwidth over all possible linear orderings of V .

The cutwidth of a graph can be computed in FPT time using the cutwidth as the parameter. In particular, there is an $2^{O(k^2)}n$ -time algorithm to compute the cutwidth k if a graph [17]. There is a connection between cutwidth and the general crossing number of graphs: Any graph $G = (V, E)$ with cutwidth k requires at least $\frac{k^2}{1176} - \sum_{v \in V} \left(\frac{\deg(v)}{4}\right)^2$ crossings in any drawing [8].

Since OSCM is NP-hard even for star forests of degree 4 [32], which have cutwidth 2, it is paraNP-hard if parameterized by the cutwidth. However, in the input to OSCM, the order of A is fixed, and the instances of the NP-hardness proof come with orders of A that do not admit a constant cutwidth, that is, there is no linear order on the vertices of the constructed graphs where the order of A is the same as in the input and the cutwidth is constant. Hence, we are interested in studying OSCM for graphs where the cutwidth remains small even if the order of A has to remain the same as in the input.

3 The Setup of PACE 2024

As already mentioned before, this year's challenge featured three tracks. The *exact* and *parameterized* track both required the computation of an optimal solution and gave different guarantees about the structure of the input instances. In the *heuristic* track, no guarantees were made about the structure of the instances and participants were tasked with computing the best (but not necessarily optimal) layout they can find within a more limited time frame.

3.1 The Exact Track

The task of this track was to compute an optimal solution of OSCM for 200 graphs, 100 of which were public and 100 of which were not known by the participants and were only presented to the solver during the evaluation in a compartmentalized judge system. For each of the graphs, the solver had a time limit of *30 minutes* and a memory limit of *8 GB* to output a solution. All instances were guaranteed to have a “not too large” number of crossings in an optimal solution (detailed information about this can be found in Section 3.5).

The organizers of the competition encouraged submissions that implement provably optimal algorithms, however, this was not a formal requirement. The exact rule stated on the website was

Submissions should be based on provably optimal algorithms, however, this is not a formal requirement. Submissions that output an incorrect solution or a solution that is known to be non-optimal will be disqualified. Besides dedicated algorithms, we also encourage submissions based on other paradigms such as SAT, MaxSAT, or ILPs.

The requirement of outputting optimal solutions extends to instances that were not included in the set of the 200 evaluation instances. In the exact track 7 solvers were disqualified, 5 on the basis of outputting wrong answers for one or more of the 200 evaluation instances and 2 due to subsequently found counterexample instances.

Submissions of this track were ranked by the *number of solved instances*. In case of a tie, the winner was determined by the *total time spent on the solved instances*. In particular, there was no need to abort a “hopeless” run early.

3.2 The Heuristic Track

In this track, the solvers were tasked with computing a good solution quickly. The solvers were run on each instance for *5 minutes* and received the Unix signal SIGTERM afterwards. When receiving this signal, the solver had to output a valid layout in the defined solution format immediately to the standard output and terminate. If the program did not halt in a reasonable time after receiving the signal, it was stopped via SIGKILL and the instance was counted as time limited exceeded. The memory limit for this track was *8 GB* as well. For this track, solutions did not have to be optimal.

Submissions were ranked by the sum over all instances of

$$\frac{\# \text{ crossings in solver layout}}{\text{smallest } \# \text{ crossings known to the PC}}$$

Note that the “smallest number of crossings in any layout known to the PC” may not be optimal, i. e., may be larger than the number of crossings in an optimal solution.

3.3 The Parameterized Track

This track had the same rules as the Exact Track. However, the instances here could require a large number of crossings, but they had small cutwidth: there is an ordering of the vertices of the graph such that every cut obtained by partitioning the vertices into earlier and later subsets of the ordering is crossed by at most “a small number” of edges. Such an ordering was provided in the input. Note that in this ordering the vertices of *A* and *B* were generally interleaved, but the order of the vertices of *A* (the fixed side) was the same as in the problem instance.

3.4 Internal Solver

One of the most challenging aspects of creating the benchmark set was to strike a balance between sufficiently difficult instances and the need to solve them to optimality in order to judge the exact and parameterized track. The organizers used an ILP solver to answer this question. This solver was implemented based on the formulation of Jünger and Mutzel [23].

Specifically, let A and B be the two partite sets of the given bipartite graph. For every pair of vertices $v_i, v_j \in B$ we create a binary variable $t_{i,j}$, which should be true if and only if v_i appears before v_j in the final order of B . Since this order is linear it is of course transitive. This can be enforced by adding for every $i < j < k$ the inequalities $0 \leq t_{i,j} + t_{j,k} - t_{i,k} \leq 1$, i.e., transitivity constraints. The number of crossings between the edges incident to v_i and the edges incident to v_j is only dependent on the relative order of v_i and v_j and can easily be precomputed. Let $c_{i,j}$ ($c_{j,i}$) be this number if v_i appears before (after) v_j in the final order of B . Then we simply minimize $\sum_{i < j} (c_{i,j}t_{i,j} + c_{j,i}(1 - t_{i,j}))$.

Since the number of transitivity constraints can grow quite large, the internal solver of the organizers added them on demand using callbacks whenever a solver found a new solution. The solver was initialized without any transitivity constraints. If a new solution is found, the binary variables are used to define a directed graph on all vertices of B , i.e., an edge is added from v_i to v_j if $t_{i,j}$ is true otherwise the edge in the opposite direction is added. If this graph contains any cycles, we obtain all chordless cycles (which are necessarily triangles) and add the transitivity constraint for the three involved vertices.

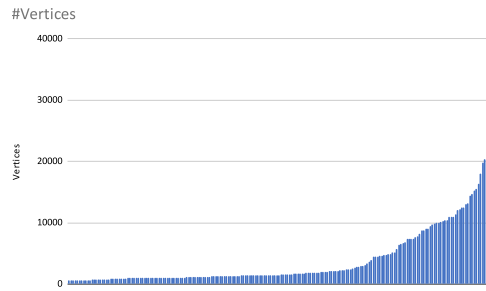
3.5 Benchmark Set

The fourth aim of the PACE challenge was to *produce universally accessible libraries of implementations and repositories of benchmark instances*. While the first part of this aim was exactly what we expected from the participants, it was the duty of the program committee to produce the benchmark instances. The properties of the benchmark instances we strived were that

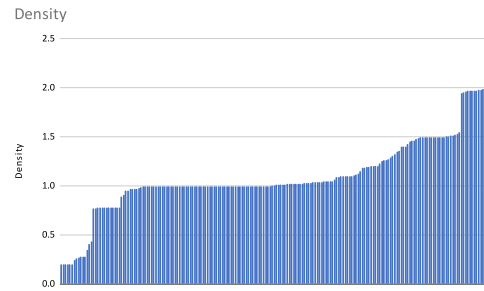
1. the benchmark instances should be heterogeneous,
2. they vary in size and difficulty and
3. it remains a challenging benchmark after the challenge.

The reason for the first criterion was that we wanted to evaluate the overall performance of the approaches developed by the participants (and not the performance on, say, a specific graph class). The goal of the second criterion was to make the challenge interesting and fun. We wanted a benchmark set in which every participant can solve at least a few instances, which should especially encourage student teams to participate as well. The medium instances were the ones that were meant to distinguish the quality of the various solvers, and the hard instances ensured that the tracks which require optimal solutions could be judged based on the number of solved instances. Moreover, we wanted to create a test set which contained instances hard enough to remain interesting after the challenge, instead of one that is simply “solved” after the competition. We expected that these hard instances are barely solvable by solvers developed in the time span of the competition and, thus, leave room for further research.

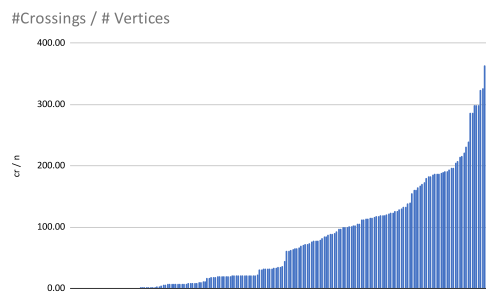
We created generators for several graph classes, including uniform random (planar) graphs, cycles, paths, complete bipartite graphs, stars, matchings, trees, lobsters, (double-)caterpillars, grids, quadrangulations, (partial) k -trees, wheels, disk intersection graphs, interval bigraphs, (circular) ladders, hypercubes, co-graphs, intersection graphs, bipartite permutation graphs, and graphs with small vertex cover / cutwidth / neighborhood diversity.



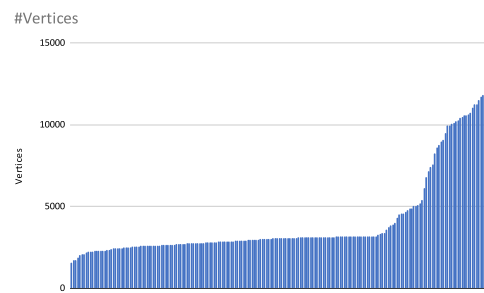
(a) Distribution of #vertices in the exact benchmark set.



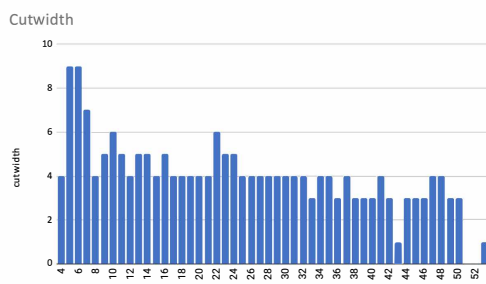
(b) Distribution of edge density in the exact benchmark set.



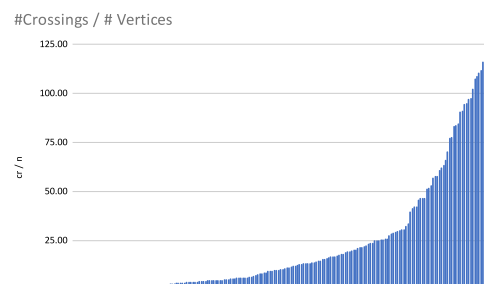
(c) Distribution of crossing density in the exact benchmark set.



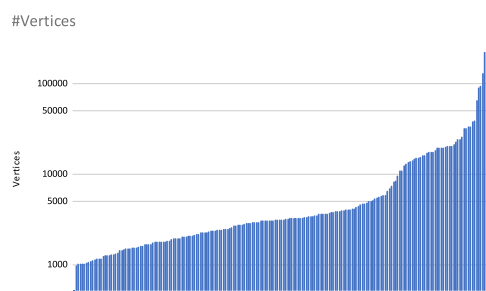
(d) Distribution of #vertices in the parameterized benchmark set.



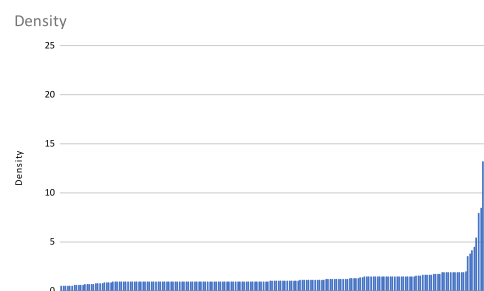
(e) Number of instances by cutwidth in the parameterized benchmark set.



(f) Distribution of crossing density in the parameterized benchmark set.



(g) Distribution of #vertices in the heuristic benchmark set (log scale).



(h) Distribution of edge density in the heuristic benchmark set.

■ **Figure 3** Details about the benchmark sets in the three tracks.

We also included (variations of) known examples from the literature that common heuristics perform badly on. To further increase variety, we also added options to randomly permute one or both bipartitions and to glue instances together. Many generators are based on Networkx [21]. The code can be found online¹.

To find a good balance in the difficulty of the benchmark, we created about 15,000 instances of various sizes from each of these graph classes and tried to solve them with our solver. For the exact and parameterized track, we only selected instances for which we knew the optimum solution.

For the exact track, we decided to use between 5 and 15 instances from each graph class. We used 60 instances that our solver could solve within 10 minutes, 50 instances that it could solve in under 20 minutes, 40 instances that it could solve in under 30 minutes, and 50 instances that required more time. While general instances can require $\Omega(m^2) \in \Omega(n^4)$ crossings, we selected instances where the *crossing density* (i.e., the number of crossings required divided by the number of vertices) does not get too large; see Figure 3c. The graphs had between 560 and 20,000 vertices (with one exception that has 32,691 vertices) and edge density between 0.2 and 2.5; see Figures 3a and 3b.

For the heuristic track, we used similar instances, but made them much larger; see Figures 3g and 3h. The graphs had up to 262,124 vertices and 1,114,112 edges.

For the parameterized track, we also used only instances for which we knew the optimum solution. We generated instances with bounded cutwidth by two random sampling approaches. All but two instances had cutwidth at most 50; see Figure 3e. The graphs had between 1552 and 13,674 vertices and required between $0.08n$ and $123.78n$ crossings.

3.6 Available Tools

Mini Test Set

As a first set of instances and to get development for the participants off the ground, we provided a set of mini test instances². We also provided the solutions to these test instances³. An impression of these instances can be seen in Figure 4.

Verifier

We provided a Python-based tool to verify a solution and test a given solver against a set of tests. This verifier was provided as a python package on the website⁴ and was continuously updated throughout the challenge. To verify a produced solution two main algorithms were available. The first one was a simple, inefficient, but surely correct implementation that just tested for every pair of edges whether they cross. The second method and usually the default option, was an algorithm based on so-called segment trees. For comfort the verifier could not only be run on a single solution to verify it, but instead could be provided with a folder of test instances and a solver. Each instance was then solved using the provided solver and the resulting solution checked. By default the test instances used in this verifier mode were the tiny test set described above.

¹ https://pacechallenge.org/2024/instance_generator.py

² https://pacechallenge.org/2024/tiny_test_set.zip

³ https://pacechallenge.org/2024/tiny_test_set-sol.zip

⁴ <https://pacechallenge.org/2024/verifier/>

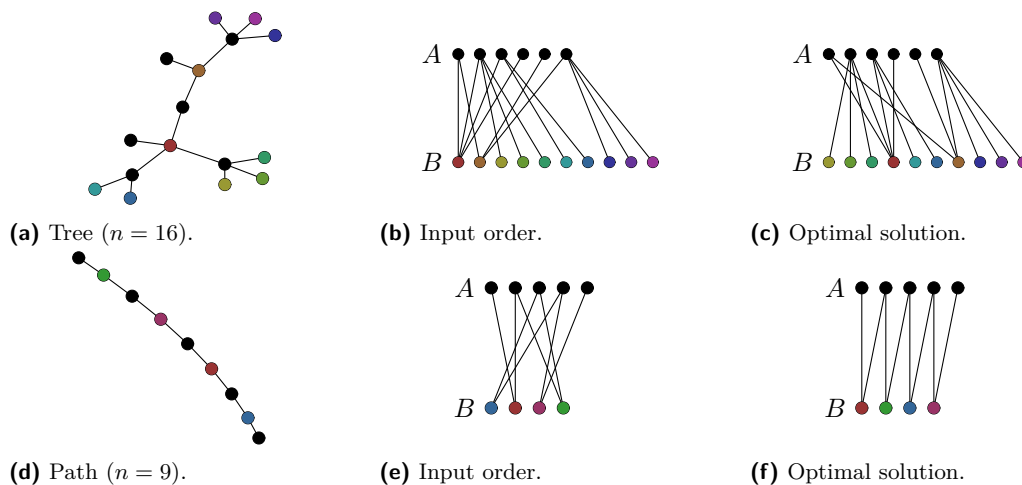


Figure 4 Two example instances from the mini test set given with a force directed layout to illustrate the structure as well as the input order and the optimal 2-layered layout. The partite set, whose order is fixed (A) is drawn in black, the vertices of the permutable set (B) are drawn in unique colors.

Visualizer

We also provided a visualizer tool to the participants.⁵ This tool displayed a given graph and solution in a graphical interface. For small and medium size instances this allowed the participants to get a graphical idea of their solutions and was hopefully helpful in finding potential improvements of their algorithms. The images in Figure 4 were created using this visualizer.

4 Participants and Results

There were 25, 32 and 17 distinct submissions to the exact, heuristic and parameterized track, respectively. Hence, in total there were 40 distinct teams with a total number of 112 participants representing four continents and 21 countries, which made this the largest PACE challenge yet. The results are listed below.

4.1 Ranking of the Exact Track

The ranking for the exact track is listed subsequently; We list the number of solved instances from the 100 private instances plus the 100 public instances as well as the total computation time used for the solved instances. Submissions marked with an “🎓” icon are student submissions after the following rules

A student is someone who is not and has not been enrolled in a PhD program before the submission deadline. A submission is eligible for a Student Submission Award if either all its authors are students, or besides student co-author(s) there is one non-student co-author that confirms, at the moment of submission, that a clear majority of conceptual and implementation work was done by the student co-author(s).

⁵ <https://pacechallenge.org/2024/visualizer/>

26:10 PACE 2024: One-Sided Crossing Minimization

The first five valid student submissions were eligible for a Student Submission Award; there were three such submissions, which received this award.

- Rank 1** [mppeg](#) solved [199](#) instances in [5682.93](#) seconds. [Link](#) 
- Authors** Michael Jünger, Paul Jünger, Petra Mutzel and Gerhard Reinelt
Affiliation University of Cologne, University of Bonn and Heidelberg University
- Rank 2** [uzl](#) solved [195](#) instances in [7692.89](#) seconds. [Link](#) 
- Authors** Max Bannach, Florian Chudigiewitsch, Kim-Manuel Klein and Marcel Wienöbst
Affiliation European Space Agency and Institute for Theoretical Computer Science, University of Lübeck
- Rank 3** [CRGone](#) solved [192](#) instances in [15520.39](#) seconds. [Link](#) 
- Authors** Alexander Dobler
Affiliation Technische Universität Wien
- Rank 4** [Guilucand](#) solved [187](#) instances in [9358.96](#) seconds. [Link](#) 
- Authors** Andrea Cracco
Affiliation Università degli Studi di Verona
- Rank 5** [crossy](#) () solved [180](#) instances in [19099.31](#) seconds. [Link](#) 
- Authors** Tobias Röhr and Kirill Simonov
Affiliation Hasso Plattner Institute, University of Potsdam
- Rank 6** [weberknecht](#) solved [164](#) instances in [21408.17](#) seconds. [Link](#) 
- Authors** Johannes Rauch
Affiliation Institute of Optimization and Operations Research, Ulm University
- Rank 7** [LUNCH](#) solved [157](#) instances in [10425.52](#) seconds. [Link](#) 
- Authors** Kenneth Langedal, Matthias Bentert, Thorgal Blanco and Pål Grønås Drange
Affiliation University of Bergen
- Rank 8** [Arcee](#) () solved [152](#) instances in [11189.13](#) seconds. [Link](#) 
- Authors** Kimon Boehmer, Lukas Lee George, Fanny Hauser and Jesse Palarus
Affiliation Université Paris-Saclay, Technical University Berlin
- Rank 9** [lcs](#) solved [136](#) instances in [1186.94](#) seconds. [Link](#) 
- Authors** Mohamed Mahmoud Abdelwahab, Faisal N. Abu-Khzam and Lucas Isenmann
Affiliation Lebanese American University
- Rank 10** [sherby](#) solved [135](#) instances in [1683.19](#) seconds. [Link](#) 
- Authors** Manuel Lafond, Alitzel López Sánchez and Bertrand Marchand
Affiliation Department of Computer Science, University of Sherbrooke
- Rank 11** [U_OCM](#) solved [121](#) instances in [7907.2](#) seconds. [Link](#) 
- Authors** Mert Biyikli, Kathrin Hanauer, Sophia Heck, Lukas Krumpeck, Lara Ost, Tobias Prisching, Ole Schlüter, Matej Vedak and Maximilian Vötsch
Affiliation Faculty of Computer Science, University of Vienna and Faculty of Physics, University of Vienna
- Rank 12** [roundabout](#) solved [109](#) instances in [564.63](#) seconds. [Link](#) 
- Authors** Emmanuel Arrighi and Petra Wolf
Affiliation EnsL, Univ Lyon, UCBL, CNRS, Inria in Lyon and LaBRI, CNRS, Université de Bordeaux
- Rank 13** [HWoydt](#) solved [75](#) instances in [12.75](#) seconds. [Link](#) 
- Authors** Henning Martin Woydt
Affiliation Heidelberg University

Rank 14 [GOAT](#) solved 74 instances in 725.36 seconds. [Link](#) 

Authors Patrik Drbal, Michal Dvořák, Dušan Knop, Jozef Koleda, Jan Pokorný and Ondřej Suchý

Affiliation Czech Technical University in Prague

Rank 15 [trex-ufmg-ilp](#) solved 41 instances in 3918.34 seconds. [Link](#) 

Authors Luis Henrique Gomes Higino, Kaio Henrique Masse Vieira, Alan Prado, Guilherme de Castro Mendes Gomes, Laila Melo Vaz Lopes, Gabriel Ubiratan Barreto Pereira de Oliveira, Gabriel Lucas Costa Martins, Heitor Gonçalves Leite, Matheus Torres Prates, Gabriel Vieira and Vinicius Fernandes dos Santos

Affiliation Departamento de Ciência da Computação, Universidade Federal de Minas Gerais

Rank 16 [studentgroupfuberlin](#)  solved 28 instances in 12844.18 seconds. [Link](#) 

Authors Garvin Konopka, Colin Alexander Voigt and Joshua Alexander Hanheiser

Affiliation Freie Universität Berlin

The details of the remaining 9 teams, which were either disqualified or did not solve any instance correctly, can be found on the PACE website. Submission [mjdv](#) solved 157 instances in 5000.98 seconds, but was disqualified due to suboptimal output on one instance due to a bug in their code.


4.2 Methods used by the Winners of the Exact Track















The **Exact Track winner** [mpeg](#) [24] reformulate the problem of the challenge into a linear ordering problem, which they solve using a branch & cut approach [19, 20]. Their method formulates the problem as an integer linear program, which uses binary variables to indicate the order of any ordered pair of vertices. Transitivity constraints enforce that there are no cycles in the ordering implied by a valid solution to the program. In the original formulation, preventing cycles of length three is sufficient, however the approach of [mpeg](#) utilizes multiple methods to speed up their approach, by reducing the size of the ILP. As a result additional cycles have to be prevented via constraint. The speed-up methods include among others the decomposition of instances into connected components and fixing any relative order between vertices. For instances where it is feasible, they additionally compute an initial solution using the “Kernigham-Lin 2” heuristic [35].






















The **runner-up** [uzl](#) [4] observed that the ILP formulation is equivalent to the WEIGHTED FEEDBACK ARC SET problem, which can also be expressed as a HITTING SET problem, where all cycles are sets. This team’s approach is based on existing formulations for WEIGHTED FEEDBACK ARC SET [1, 19]. Initially only some cycle constraints are added. After finding a solution to a relaxation of their model, a heuristic is used to identify new cycle constraints until all cycles are removed.

The **third-place** [CRGone](#) [10] again uses an ILP formulation after employing some reduction rules to decrease the instance size. This is, e.g., done by contracting vertices with the same neighborhood or (similar to the winning team) fixing ordering variables, if one of the two relative orders creates no crossings. The model is initialized without any transitivity constraints between binary variables, which are separated in a predefined order, preferring constraints for vertices whose neighbors do not interleave.








4.3 Ranking of the Heuristic Track

The ranking for the heuristic track is listed subsequently; We list the score for the 100 private instances plus the 100 public instances, computed as described above, as well as the total computation time. The larger the score, the better. Submissions marked with an “” icon are student submissions (using the same rules as above) of which the top five obtained a Student Submission Award.

- Rank 1** [CIMAT_Team](#) got a score of [199.99996](#) in [59236.67](#) seconds. [Link](#) 
- Authors** Carlos Segura, Lázaro Lugo, Gara Miranda and Edison David Serrano Cárdenas
Affiliation Area de Computación, Centro de Investigación en Matemáticas (CIMAT) in Mexico, Departamento de Ingeniería Informática y de Sistemas, Universidad de La Laguna and Area de Matemáticas Aplicadas, Centro de Investigación en Matemáticas (CIMAT) in Mexico
- Rank 2** [LUNCH](#) got a score of [199.99994](#) in [80545.43](#) seconds. [Link](#) 
- Authors** Kenneth Langedal, Matthias Bentert, Thorgal Blanco and Pål Grønås Drange
Affiliation University of Bergen
- Rank 3** [Martin_J_Geiger](#) got a score of [199.99983](#) in [41662.87](#) seconds. [Link](#) 
- Authors** Martin Josef Geiger
Affiliation University of the Federal Armed Forces Hamburg
- Rank 4** [Arcee](#)  got a score of [199.9998](#) in [38339.44](#) seconds. [Link](#) 
- Authors** Kimon Boehmer, Lukas Lee George, Fanny Hauser and Jesse Palarus
Affiliation Université Paris-Saclay, Technical University Berlin
- Rank 5** [guilhermefonseca](#) got a score of [199.99978](#) in [26336.25](#) seconds. [Link](#) 
- Authors** Guilherme D. da Fonseca
Affiliation LIS, Aix-Marseille Université
- Rank 6** [Bob](#) got a score of [199.99978](#) in [27380.75](#) seconds. [Link](#) 
- Authors** Sergey Pupyrev
Affiliation Menlo Park
- Rank 7** [uzl](#) got a score of [199.9996](#) in [80644.98](#) seconds. [Link](#) 
- Authors** Max Bannach, Florian Chudigiewitsch, Kim-Manuel Klein and Marcel Wienöbst
Affiliation European Space Agency and Institute for Theoretical Computer Science, University of Lübeck
- Rank 8** [slimmer](#) got a score of [199.99865](#) in [54761.13](#) seconds. [Link](#) 
- Authors** Steffen Limmer and Nils Einecke
Affiliation Honda Research Institute Europe GmbH
- Rank 9** [UAIC_OCM](#)  got a score of [199.99735](#) in [56047.38](#) seconds. [Link](#) 
- Authors** Andrei Arhire, Eugen Croitoru, Matei Chiriac and Alex Dumitrescu
Affiliation Alexandru Ioan Cuza University of Iași
- Rank 10** [axs](#)  got a score of [199.99037](#) in [59409.74](#) seconds. [Link](#) 
- Authors** Chenghao Zhu, Yi Zhou and Bo Peng
Affiliation University of Electronic Science and Technology of China and Southwestern University of Finance and Economics Chengdu
- Rank 11** [weberknecht](#) got a score of [199.97621](#) in [6187.64](#) seconds. [Link](#) 
- Authors** Johannes Rauch
Affiliation Institute of Optimization and Operations Research, Ulm University

- Rank 12 [tlopez](#)  got a score of [199.93344](#) in [80556.06](#) seconds. [Link](#) 
- Authors** Toan Lopez and Florian Sikora
Affiliation Student of Université Paris Dauphine
- Rank 13 [KongQi](#)  got a score of [199.66556](#) in [38664.61](#) seconds. [Link](#) 
- Authors** Qi Kong, Zhouxing Su and Zhipeng Lü
Affiliation Huazhong University of Science and Technology
- Rank 14 [heiCross](#)  got a score of [199.46119](#) in [80480.59](#) seconds. [Link](#) 
- Authors** Adil Chhabra, Marlon Dittes, Alvaro Garmendia, Ernestine Großmann, Tomer Haham, Shai Peretz, Henrik Reinstädler, Antonie Wagner and Henning Woydt
Affiliation University of Heidelberg
- Rank 15 [LOPP](#)  got a score of [198.93312](#) in [80720.08](#) seconds. [Link](#) 
- Authors** Arijeet Pramanik, Rishabh Dev, Vimal Narassimane and Srinibas Swain
Affiliation Department of Computer Science and Engineering, IIT Guwahati
- Rank 16 [GAON](#)  got a score of [198.88211](#) in [80720.77](#) seconds. [Link](#) 
- Authors** Rishabh Dev, Arijeet Pramanik, Vimal Narassimane and Srinibas Swain
Affiliation Department of Computer Science and Engineering, IIT Guwahati
- Rank 17 [ericweidner](#)  got a score of [198.79449](#) in [4961.61](#) seconds. [Link](#) 
- Authors** Carolin Rehs and Eric Weidner
Affiliation Technical University of Dortmund
- Rank 18 [KUL-TW](#) got a score of [198.56896](#) in [19684.34](#) seconds. [Link](#) 
- Authors** Tony Wauters and Fabien Nießen
Affiliation NUMA, Department of Computer Science, KU Leuven
- Rank 19 [DRIP](#)  got a score of [198.30131](#) in [80720.22](#) seconds. [Link](#) 
- Authors** Unknown – no solver description submitted
Affiliation Unknown – no solver description submitted
- Rank 20 [lmsrusso](#) got a score of [198.19412](#) in [77464.5](#) seconds. [Link](#) 
- Authors** Luís M. S. Russo
Affiliation INESC-ID and Department of Computer Science and Engineering, Instituto Superior Técnico, Universidade de Lisboa
- Rank 21 [GOAT](#) got a score of [196.02268](#) in [235.44](#) seconds. [Link](#) 
- Authors** Patrik Drbal, Michal Dvořák, Dušan Knop, Jozef Koleda, Jan Pokorný and Ondřej Suchý
Affiliation Czech Technical University in Prague
- Rank 22 [NV_OCM](#) got a score of [194.69549](#) in [5519.86](#) seconds. [Link](#) 
- Authors** André Nusser and Juliette Vlieghe
Affiliation CNRS, Inria Center at Université Côte d’Azur and Technical University of Denmark
- Rank 23 [HCPS42](#) got a score of [192.17673](#) in [2172.91](#) seconds. [Link](#) 
- Authors** Temirkhan Zimanov
Affiliation Higher School of Economics
- Rank 24 [asdf](#) got a score of [188.77488](#) in [27596.93](#) seconds. [Link](#) 
- Authors** Unknown – no solver description submitted
Affiliation Unknown – no solver description submitted
- Rank 25 [simonhol](#) got a score of [181.14588](#) in [4036.8](#) seconds. [Link](#) 
- Authors** Simon Hol
Affiliation Utrecht University

26:14 PACE 2024: One-Sided Crossing Minimization

- Rank 26** [U_OCM](#) got a score of [181.05501](#) in [80636.53](#) seconds. [Link](#) 
- Authors** Mert Biyikli, Kathrin Hanauer, Sophia Heck, Lukas Krumpeck, Lara Ost, Tobias Prisching, Ole Schlüter, Matej Vedak and Maximilian Vötsch
Affiliation Faculty of Computer Science, University of Vienna and Faculty of Physics, University of Vienna
- Rank 27** [iitg](#) got a score of [180.18229](#) in [480.38](#) seconds. [Link](#) 
- Authors** Sahaj Gupta, Swati Nanda Gupta, Sampriti Patel and Srinibas Swain
Affiliation Department of Computer Science and Engineering, IIIT Guwahati
- Rank 28** [Guilucand](#) got a score of [172.30134](#) in [16750.69](#) seconds. [Link](#) 
- Authors** Andrea Cracco
Affiliation Università degli Studi di Verona
- Rank 29** [DumbAndDumber](#) got a score of [165.28777](#) in [1904.8](#) seconds. [Link](#) 
- Authors** Kristoffer Sandvang and Mateusz Filipowski
Affiliation Student at the University of Copenhagen
- Rank 30** [roundabout](#) got a score of [163.44052](#) in [32077.4](#) seconds. [Link](#) 
- Authors** Emmanuel Arrighi and Petra Wolf
Affiliation EnsL, Université Lyon, UCBL, CNRS, Inria in Lyon and LaBRI, CNRS, Université de Bordeaux
- Rank 31** [oscmpp](#) got a score of [112.20293](#) in [54079.36](#) seconds. [Link](#) 
- Authors** Sahaj Gupta, Swati Nanda Gupta, Sampriti Patel and Srinibas Swain
Affiliation Department of Computer Science and Engineering, IIIT Guwahati
- Rank 32** [WINTER](#) got a score of [109.28239](#) in [57113.3](#) seconds. [Link](#) 
- Authors** Sahaj Gupta, Swati Nanda Gupta, Sampriti Patel and Srinibas Swain
Affiliation Department of Computer Science and Engineering, IIIT Guwahati

4.4 Methods used by the Winners of the Heuristic Track


The **Parameterized Track** winner **CIMAT_Team** [36] used a first generation memetic algorithm with explicit diversity management. They initially generated a population of random solutions. They used iterated local search to improve these solutions. Their approach applied cycle-based crossover and applied a Best-Non-Penalized survivor selection strategy, which adjusts over time to favor exploration early and exploitation later, thereby promoting diversity. For larger instances, their method shifted to a more direct application of iterated local search. They used a greedy initialization strategy based on scoring vertices to manage complexity and memory efficiently.














Similar to team **uzl** in the exact track, the **runner-up LUNCH** [29] reduced OSCM to the **WEIGHTED FEEDBACK ARC SET** problem. They observed that edges between strongly connected components can be deleted as they are not part of any cycles, and that strongly connected components can be solved independently. They proved that several edges can be ignored and thus managed to create sparser instances that are sufficient to quickly find strongly connected components. They used a dynamic program [30] to solve components of at most 20 vertices optimally. For larger components, they first used greedy improvements, then an adjusted cutting technique by Park and Akers [34].

The **third-place Martin_J_Geiger** [16] used iterated local search and variable neighborhood search to find good solutions. They first applied the reduction rules by Dujmović et al. [12] and then used the barycenter heuristic to find an initial solution. They iteratively improved the solution with small improving moves until they reached a local optimum; to

this end, they either moved a single node or a block of up to 5 subsequent nodes to different positions. To escape local optima, they reversed a subset of up to 20% of the permutation and continued their search from there.

4.5 Ranking of the Parameterized Track

The ranking for the parameterized track is listed subsequently; We list the number of solved instances from the 100 private instances plus the 100 public instances as well as the total computation time. Three teams were disqualified because instances were found for which their solver did return a suboptimal solution. Submissions marked with an “” icon are student submissions after the same rules as above. There were two such submissions, which received the Student Submission Award.

- Rank 1** [LUNCH](#) solved 200 instances in 5.15 seconds. [Link](#) 
- Authors** Kenneth Langedal, Matthias Bentert, Thorgal Blanco and Pål Grønås Drange
Affiliation University of Bergen
- Rank 1** [mjdv](#) solved 200 instances in 10.37 seconds. [Link](#) 
- Authors** Ragnar Groot Koerkamp and Mees de Vries
Affiliation ETH Zurich and Unaffiliated in The Netherlands
- Rank 3** [mjpeg](#) solved 200 instances in 25.22 seconds. [Link](#) 
- Authors** Michael Jünger, Paul Jünger, Petra Mutzel and Gerhard Reinelt
Affiliation University of Cologne, University of Bonn and Heidelberg University
- Rank 4** [Arcee](#)  solved 200 instances in 28.54 seconds. [Link](#) 
- Authors** Kimon Boehmer, Lukas Lee George, Fanny Hauser and Jesse Palarus
Affiliation Université Paris-Saclay, Technical University Berlin
- Rank 5** [crossy](#)  solved 200 instances in 34.98 seconds. [Link](#) 
- Authors** Tobias Röhr and Kirill Simonov
Affiliation Hasso Plattner Institute, University of Potsdam
- Rank 6** [uzl](#) solved 200 instances in 60.49 seconds. [Link](#) 
- Authors** Max Bannach, Florian Chudigiewitsch, Kim-Manuel Klein and Marcel Wienöbst
Affiliation European Space Agency and Institute for Theoretical Computer Science, University of Lübeck
- Rank 7** [roundabout](#) solved 200 instances in 121.23 seconds. [Link](#) 
- Authors** Emmanuel Arrighi and Petra Wolf
Affiliation EnsL, Université Lyon, UCBL, CNRS, Inria in Lyon and LaBRI, CNRS, Université de Bordeaux
- Rank 8** [CRGone](#) solved 200 instances in 125.07 seconds. [Link](#) 
- Authors** Alexander Dobler
Affiliation Technische Universität Wien
- Rank 9** [Guilucand](#) solved 200 instances in 162.07 seconds. [Link](#) 
- Authors** Andrea Cracco
Affiliation Università degli Studi di Verona
- Rank 10** [weberknecht](#) solved 200 instances in 287.41 seconds. [Link](#) 
- Authors** Johannes Rauch
Affiliation Institute of Optimization and Operations Research, Ulm University
- Rank 11** [sherby](#) solved 199 instances in 20.84 seconds. [Link](#) 
- Authors** Manuel Lafond, Alitzel López Sánchez and Bertrand Marchand
Affiliation Department of Computer Science, University of Sherbrooke

26:16 PACE 2024: One-Sided Crossing Minimization

Rank 12 [trex-ufmg](#) solved **198** instances in **14848** seconds. [Link](#) 

Authors Luis Henrique Gomes Higino, Kaio Henrique Masse Vieira, Alan Prado, Guilherme de Castro Mendes Gomes, Laila Melo Vaz Lopes, Gabriel Ubiratan Barreto Pereira de Oliveira, Gabriel Lucas Costa Martins, Heitor Gonçalves Leite, Matheus Torres Prates, Gabriel Vieira and Vinicius Fernandes dos Santos

Affiliation Departamento de Ciência da Computação, Universidade Federal de Minas Gerais

Rank 13 [narekb95](#) solved **163** instances in **13082.35** seconds. [Link](#) 

Authors Narek Bojikian

Affiliation Humboldt University of Berlin

4.6 Methods used by the Winners of the Parameterized Track

The **Parameterized Track winner LUNCH** [29] started with the same approach as in their second place submission to the heuristic track. After running the heuristic for each large component to get an upper bound, they solved a MaxSAT instance to optimally eliminate all cycles of length at most 4. If this led to an acyclic instances or to a solution that had the same cost as the upper bound, the algorithm terminated. Otherwise, they removed these edges, found new cycles with a DFS traversal, and repeated the previous steps.

The **runner-up mjdv** [27] did not rely on ILP or SAT formulations and instead used a branch-and-bound algorithm. They generalized reduction rules from Dujmović et al. [12] to find pairs of vertices in B where one must lie to the left of the other in any optimal solution, or pairs that are placed directly next to each other in some optimal solution. They also generalized a reduction rule to find a vertex that lies at the leftmost position in some optimal solution. As a lower bound, they calculated for each pair of vertices in B the minimum number of crossings required between their incident edges in any drawing [11, 15]. They then fixed vertices in the solution from left to right, keeping track of the number of crossings involved with the fixed vertices. It uses the reduction rules explained above to fix pairs of vertices and find the next vertex to add to the prefix, which is then inserted at the optimal position. Caching previously found lower bounds speeds up the process.

The **third-place mpeg** [24] used the same solver as in the exact track.

5 PACE Organization

The program committee of PACE 2024 consisted of Philipp Kindermann (Universität Trier, chair), Fabian Klute (UPC Barcelona) and Soeren Terziadis (Eindhoven University of Technology). During the competition, the members of the steering committee were:

- (since 2023) Max Bannach (European Space Agency)
- (since 2023) Sebastian Berndt (Universität zu Lübeck)
- (since 2016) Holger Dell (Goethe University Frankfurt and IT University of Copenhagen)
- (since 2016) Bart M. P. Jansen (chair) (Eindhoven University of Technology)
- (since 2020) Lukasz Kowalik (University of Warsaw)
- (since 2021) André Nichterlein (Technical University of Berlin)
- (since 2022) Christian Schulz (Universität Heidelberg)
- (since 2020) Manuel Sorge (Technische Universität Wien)

The Program Committee of PACE 2025 will be chaired by Sebastian Siebertz und Mario Grobler (both University of Bremen).

6 Conclusion, Reflection and Future Editions of PACE

We thank all the participants for their impressive work, vital contributions, and patience in case of technical issues. The organizers especially thank the participants who presented their work at IPEC 2024 in the poster session or during the award ceremony. We are pleased about the large number of diverse participants and hope the PACE challenge will continue to build bridges between theory and practice. We welcome anyone interested to add their name to the mailing list on the PACE website to receive updates and join the discussion.

The Good – What went well

OSCM was a good choice as the problem for the competition: the problem is simple to understand, requires no background knowledge, and the solutions are easy to verify. We used a Zulip server for direct communication with and between the participants. This server was very active, it made it much easier for us to communicate updates with the participants, and gave them the opportunity to discuss issues and help each other. We had a very large number of participants, much more than in the previous years. While many of the approaches were similar in nature, each team found different tricks to reduce instance sizes or to overcome obstacles. The poster exhibition during IPEC where the winning teams shared the knowledge they obtained about the problem during the contest was well attended and sparked several hours worth of discussion. Overall, there was a very friendly and helpful atmosphere between contestants. The exact and heuristic tracks were very successful with many different submissions. The participants were also very quick in finding any technical mistakes that the program committee had made in the provided tools and instances.

The Bad – What could we have done better

Since we guaranteed the exact instances to have not too large crossing numbers, we could only use instances for the benchmark sets where we knew the optimum solution. But since our internal solver turned out to be very slow compared to the submissions we received, this limited the difficulty for the instances we could provide. For example, our solver needed in total 567 hours to solve the 200 instances of the whole parameterized benchmark set, while the best submission could solve all of them in merely 5 seconds. In the future, we suggest that there should also be graphs where the optimum solution is unknown, even if that means that one cannot give guarantees on the solution. It might make sense to follow the example of the SAT competition, where each participating team also has to submit a small number of interesting benchmark instances that are then added to the evaluation set.

Because of our slow solver, we severely overestimated the difficulty of the parameterized benchmark set. Hence, the parameterized track was more akin to an algorithm engineering competition where the participants were fighting to scrape off milliseconds from their running times instead of figuring out how to solve larger instances. To the best of our knowledge, no submission in this track even used the property that the graphs have small cutwidth. Overall, the parameterized track did not work out well this time and should be strongly revised before running it again; there should at least be much larger graphs, larger parameters and/or less allowed computation time.

The public repository that we set up for the community to share interesting instances with each other was unused – there have been no pushes except by the program committee. After the final evaluation and publishing the private benchmark sets, one participant found that 5 of the instances had multi-edges. All input graphs were supposed to be simple; this

was due to a bug in our graph generators. We had to fix these instances, rerun all solvers on the fixed versions and change the ranking after publishing the results, which was unfortunate for teams that dropped in the rankings.

The Ugly – What did we struggle with

We struggled a bit with the tight timeline, as we underestimated the huge computation times required to find solutions for the benchmark sets. We had access to a 92-core server that was running non-stop for 4 months. It was very tough for us to predict how hard instances are and to find interesting instances that fit all constraints to make them interesting: for example, they should have not too few edges, not require too many crossings, and common heuristics should not immediately find optimum solutions. Many of these could only be checked after finding their optimum solution. Thus, we had to create a huge number of potential instances, many of which turned out to be unusable.

The servers by Optil.io were overloaded by the many submissions. During the last few days, participants had to wait for many hours until they received a result from their submitted solvers. This also had the effect that the running times varied a lot; submitting the same solver twice could lead to completely different timing results. Unfortunately, we could not find an alternative, as the required server load is too large for non-commercial options. Hence, we had to evaluate all solvers on our own server after the submission deadline. This also took a long time: to ensure that each run receives the exact same resources, we could only solve 20 instances in parallel, each of which could take a bit more than 30 minutes, so we needed up to 5 hours computation for each each of the 75 submissions. While some solvers were easy to compile and run, others required more time: 15 solvers did not immediately compile or run without major issues and required help from the teams. To reduce the workload for the program committee in the future, one option would be to let the participants submit docker images that should remove these issues.

Finally, one large point of discussion were the rules for the exact track, in particular that it was not a formal requirement that submissions have to be based on provably optimal algorithms. The reason for this non-requirement was to encourage new techniques even without finding a theoretical proof for them. However, this led to two submissions that were based on heuristic solvers that just happened to solve also all instances of the exact benchmark set optimally. The submissions were disqualified due to subsequently found counterexample instances. For the future, these rules should be revised; either there should be more strict requirements or the community should get the option to review submitted solvers and find counterexamples before the rankings are published.

PACE 2025

We look forward to the next edition, which will focus on DOMINATING SET and HITTING SET and will be chaired by Sebastian Siebertz and Mario Grobler. Detailed information will be posted on the website of the competition at pacechallenge.org.

References

- 1 Ali Baharev, Hermann Schichl, Arnold Neumaier, and Tobias Achterberg. An exact method for the minimum feedback arc set problem. *ACM J. Exp. Algorithmics*, 26, April 2021. doi:10.1145/3446429.
- 2 Tomas Balyo, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors. *SAT Competition 2023: Solver, Benchmark and Proof Checker Descriptions*. Uni. Helsinki, 2023.

- 3 Max Bannach and Sebastian Berndt. PACE solver description: The PACE 2023 parameterized algorithms and computational experiments challenge: Twinwidth. In *IPEC*, volume 285 of *LIPICs*, pages 35:1–35:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.IPEC.2023.35.
- 4 Max Bannach, Florian Chudigiewitsch, Kim-Manuel Klein, and Marcel Wienöbst. PACE solver description: UzL exact solver for one-sided crossing minimization. In *IPEC*, volume 321 of *LIPICs*, pages 28:1–28:4. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024.
- 5 Édouard Bonnet and Florian Sikora. The PACE 2018 parameterized algorithms and computational experiments challenge: The third iteration. In *IPEC*, volume 115 of *LIPICs*, pages 26:1–26:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.IPEC.2018.26.
- 6 Holger Dell, Thore Husfeldt, Bart M. P. Jansen, Petteri Kaski, Christian Komusiewicz, and Frances A. Rosamond. The first parameterized algorithms and computational experiments challenge. In *IPEC*, volume 63 of *LIPICs*, pages 30:1–30:9. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.IPEC.2016.30.
- 7 Holger Dell, Christian Komusiewicz, Nimrod Talmon, and Mathias Weller. The PACE 2017 parameterized algorithms and computational experiments challenge: The second iteration. In *IPEC*, volume 89 of *LIPICs*, pages 30:1–30:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.IPEC.2017.30.
- 8 Hristo N. Djidjev and Imrich Vrto. Crossing numbers and cutwidths. *J. Graph Algorithms Appl.*, 7(3):245–251, 2003. doi:10.7155/JGAA.00069.
- 9 Alexander Dobler. A note on the complexity of one-sided crossing minimization of trees, 2023. arXiv:2306.15339, doi:10.48550/arXiv.2306.15339.
- 10 Alexander Dobler. PACE solver description: CRGone. In *IPEC*, volume 321 of *LIPICs*, pages 29:1–29:4. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024.
- 11 Vida Dujmovic and Sue Whitesides. An efficient fixed parameter tractable algorithm for 1-sided crossing minimization. *Algorithmica*, 40(1):15–31, 2004. doi:10.1007/s00453-004-1093-2.
- 12 Vida Dujmović, Henning Fernau, and Michael Kaufmann. Fixed parameter algorithms for one-sided crossing minimization revisited. *J. Discr. Alg.*, 6(2):313–323, 2008. doi:10.1016/j.jda.2006.12.008.
- 13 M. Ayaz Dzulfikar, Johannes Klaus Fichte, and Markus Hecher. The PACE 2019 parameterized algorithms and computational experiments challenge: The fourth iteration (invited paper). In *IPEC*, volume 148 of *LIPICs*, pages 25:1–25:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.IPEC.2019.25.
- 14 Peter Eades and Sue Whitesides. Drawing graphs in two layers. *Theoret. Comp. Sci.*, 131(2):361–374, 1994. doi:10.1016/0304-3975(94)90179-1.
- 15 Peter Eades and Nicholas C. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11(4):379–403, 1994. doi:10.1007/bf01187020.
- 16 Martin J. Geiger. PACE 2024 solver description: Martin_J_Geiger. In *IPEC*, volume 321 of *LIPICs*, pages 32:1–32:4. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024.
- 17 Archontia C. Giannopoulou, Michal Pilipczuk, Jean-Florent Raymond, Dimitrios M. Thilikos, and Marcin Wrochna. Cutwidth: Obstructions and algorithmic aspects. *Algorithmica*, 81(2):557–588, 2019. doi:10.1007/S00453-018-0424-7.
- 18 Ernestine Großmann, Tobias Heuer, Christian Schulz, and Darren Strash. The PACE 2022 parameterized algorithms and computational experiments challenge: Directed feedback vertex set. In *IPEC*, volume 249 of *LIPICs*, pages 26:1–26:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.IPEC.2022.26.
- 19 Martin Grötschel, Michael Jünger, and Gerhard Reinelt. A cutting plane algorithm for the linear ordering problem. *Oper. Res.*, 32(6):1195–1220, 1984. doi:10.1287/OPRE.32.6.1195.
- 20 Martin Grötschel, Michael Jünger, and Gerhard Reinelt. Facets of the linear ordering polytope. *Math. Program.*, 33(1):43–60, 1985. doi:10.1007/BF01582010.

- 21 Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *SciPy*, pages 11–15, 2008.
- 22 Patrick Healy and Nikola S. Nikolov. Hierarchical drawing algorithms. In *Handbook on Graph Drawing and Visualization*, pages 409–453. Chapman and Hall/CRC, 2013.
- 23 Michael Jünger and Petra Mutzel. 2-layer straightline crossing minimization: Performance of exact and heuristic algorithms. In *J. Graph Algorithms Appl.*, volume 1, pages 1–25. World Scientific, 2002. doi:10.1142/9789812777638_0001.
- 24 Michael Jünger, Paul Jünger, Petra Mutzel, and Gerhard Reinelt. PACE solver description: Exact solution of the one-sided crossing minimization problem by the MPPEG team. In *IPEC*, volume 321 of *LIPICs*, pages 27:1–27:4. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024.
- 25 Leon Kellerhals, Tomohiro Koana, André Nichterlein, and Philipp Zschoche. The PACE 2021 parameterized algorithms and computational experiments challenge: Cluster editing. In *IPEC*, volume 214 of *LIPICs*, pages 26:1–26:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.IPEC.2021.26.
- 26 Yasuaki Kobayashi and Hisao Tamaki. A fast and simple subexponential fixed parameter algorithm for one-sided crossing minimization. *Algorithmica*, 72(3):778–790, 2015. doi:10.1007/s00453-014-9872-x.
- 27 Ragnar Groot Koerkamp and Mees de Vries. PACE solver description: OCMu64, a solver for one-sided crossing minimization. In *IPEC*, volume 321 of *LIPICs*, pages 35:1–35:4. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024.
- 28 Lukasz Kowalik, Marcin Mucha, Wojciech Nadara, Marcin Pilipczuk, Manuel Sorge, and Piotr Wygocki. The PACE 2020 parameterized algorithms and computational experiments challenge: Treedepth. In *IPEC*, volume 180 of *LIPICs*, pages 37:1–37:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.IPEC.2020.37.
- 29 Kenneth Langedal, Matthias Bentert, Thorgal Blanco, and Pål Grønås Drange. PACE solver description: LUNCH — linear uncrossing heuristics. In *IPEC*, volume 321 of *LIPICs*, pages 34:1–34:4. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024.
- 30 E. Lawler. A comment on minimum feedback arc sets. *IEEE Trans. Circuit Theory*, 11(2):296–297, 1964. doi:10.1109/TCT.1964.1082291.
- 31 Xiao Yu Li and Matthias F. Stallmann. New bounds on the barycenter heuristic for bipartite drawing. *Inform. Proc. Let.*, 82:293–298, 2002. doi:10.1016/S0020-0190(01)00297-6.
- 32 Xavier Muñoz, Walter Unger, and Imrich Vrto. One sided crossing minimization is NP-hard for sparse graphs. In *GD*, volume 2265 of *LNCS*, pages 115–123. Springer, 2001. doi:10.1007/3-540-45848-4_10.
- 33 Networks project, 2017. URL: <https://www.thenetworkcenter.nl>.
- 34 S. Park and S.B. Akers. An efficient method for finding a minimal feedback arc set in directed graphs. In *ISCAS*, volume 4, pages 1863–1866, 1992. doi:10.1109/ISCAS.1992.230449.
- 35 Martí Rafael and Reinelt Gerhard. Exact and heuristic methods in combinatorial optimization. *Appl. Math. Sci.*, 2022. doi:10.1007/978-3-662-64877-3.
- 36 Carlos Segura, Lázaro Lugo, Gara Miranda, and Edison David Serrano Cárdenas. PACE solver description: CIMAT_Team. In *IPEC*, volume 321 of *LIPICs*, pages 31:1–31:4. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024.
- 37 Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiro Toda. Methods for visual understanding of hierarchical system structures. *IEEE Trans. Syst., Man, and Cybernetics*, 11(2):109–125, 1981. doi:10.1109/tsmc.1981.4308636.
- 38 Markus Wallinger. *Exploring Graph-based Concepts for Balanced Information Density in Data Visualizations*. PhD thesis, TU Wien, 2024. doi:10.34726/hss.2024.124826.

PACE Solver Description: Exact Solution of the One-Sided Crossing Minimization Problem by the MPPEG Team

Michael Jünger  

University of Cologne, Germany

Paul J. Jünger  

University of Bonn, Germany

Petra Mutzel  

University of Bonn, Germany

Gerhard Reinelt  

Heidelberg University, Germany

Abstract

This is a short description of our solver `oscm` submitted by our team MPPEG to the PACE 2024 challenge both for the exact track and the parameterized track, available at <https://github.com/pauljngr/PACE2024> [9] and <https://doi.org/10.5281/zenodo.11546972> [8].

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization; Theory of computation → Mathematical optimization; Theory of computation → Parameterized complexity and exact algorithms; Human-centered computing → Graph drawings

Keywords and phrases Combinatorial Optimization, Linear Ordering, Crossing Minimization, Branch and Cut, Algorithm Engineering

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.27

Supplementary Material

Software (Source Code): <https://github.com/pauljngr/PACE2024> [9]

Software (Source Code): <https://doi.org/10.5281/zenodo.11546972> [8]

Acknowledgements The authors gratefully acknowledge the granted access to the Marvin cluster hosted by the University of Bonn.

1 Method

We apply the approach to the one-sided crossing minimization problem presented in [10]. This article is surveyed by Patrick Healy and Nikola S. Nikolov in Chapter 13.5 of the Handbook of Graph Drawing and Visualization [7] that is recommended on the PACE 2024 web page [13]. The method consists of a transformation of a one-sided crossing minimization instance to an instance of the linear ordering problem that is solved by branch&cut as introduced in [4] and [5]. We also use problem decomposition and reduction techniques as well as a heuristic for finding a good initial solution. With the required brevity, we give a rough sketch of the major details.

The instances of the PACE 2024 challenge problem consist of a bipartite graph $G = (T \cup B, E)$ and a fixed linear ordering $\pi_T = \langle t_1, t_2, \dots, t_m \rangle$ of T (“the top nodes”). In the exact track and the parameterized track, the task is to find a linear ordering π_B of $B = \{b_1, b_2, \dots, b_n\}$ (“the bottom nodes”) such that the number of edge crossings in a straight-line drawing of G with T and B on two parallel lines, following their linear orderings, is provably minimum. The NP-hardness of this task has been shown in [2].



© Michael Jünger, Paul J. Jünger, Petra Mutzel, and Gerhard Reinelt;
licensed under Creative Commons License CC-BY 4.0

19th International Symposium on Parameterized and Exact Computation (IPEC 2024).

Editors: Édouard Bonnet and Paweł Rzażewski; Article No. 27; pp. 27:1–27:4

Leibniz International Proceedings in Informatics



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

For a linear ordering π_B of B let

$$x_{ij} = \begin{cases} 1 & \text{if } b_i \text{ appears before } b_j \text{ in } \pi_B, \\ 0 & \text{otherwise.} \end{cases}$$

For $i, j \in \{1, 2, \dots, n\}$ let $c_{ii} = 0$, and for $i \neq j$ let c_{ij} denote the number of crossings between the edges incident to b_i with the edges incident to b_j if b_i appears before b_j in π_B . Then the number of crossings induced by π_B is

$$\text{cr}(\pi_B) = \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij}.$$

Since for any pair $b_i \neq b_j$ in B we have $x_{ji} = 1 - x_{ij}$, we can reduce the number of variables to $\binom{n}{2}$ and obtain

$$\text{cr}(\pi_B) = \sum_{i=1}^{n-1} \sum_{j=i+1}^n c_{ij} x_{ij} + c_{ji}(1 - x_{ij}) = \sum_{i=1}^{n-1} \sum_{j=i+1}^n (c_{ij} - c_{ji}) x_{ij} + \sum_{i=1}^{n-1} \sum_{j=i+1}^n c_{ji}.$$

For $a_{ij} = c_{ij} - c_{ji}$ we solve the *linear ordering problem* as the following binary linear program, based on the complete digraph D with node set B .

$$\begin{aligned} \text{(LO)} \quad & \text{minimize} && \sum_{i=1}^{n-1} \sum_{j=i+1}^n a_{ij} x_{ij} \\ & \text{subject to} && \sum_{\substack{(b_i, b_j) \in C: \\ i < j}} x_{ij} + \sum_{\substack{(b_i, b_j) \in C: \\ i > j}} (1 - x_{ji}) \leq |C| - 1 && \text{for all dicycles } C \text{ in } D \\ & && 0 \leq x_{ij} \leq 1 && \text{for } 1 \leq i < j \leq n \\ & && x_{ij} \text{ integral} && \text{for } 1 \leq i < j \leq n. \end{aligned}$$

If z is the optimum value of (LO), $z + \sum_{i=1}^{n-1} \sum_{j=i+1}^n c_{ji}$ is the minimum number of crossings. Notice that the classical linear ordering formulation [4, 5] uses constraints for cycles of length three only. However, in our approach we also need longer cycles, since we remove some of the arcs as we shall describe in Section 2. The constraints of (LO) guarantee that the solutions correspond precisely to all permutations π_B of B . Furthermore, it can be shown that for complete digraphs the “3-cycle constraints” are necessary in any minimal description of the feasible solutions by linear inequalities, if the integrality conditions are dropped. The NP-hardness of the problem makes it unlikely that such a complete linear description can be found. Further classes of inequalities with a number of members exponential in n that must be present in a complete linear description of the feasible set, are known, and some of them can be exploited algorithmically. Indeed, small Möbius-ladder constraints, the one shown in Figure 3 of [4], as well as the same in which all arcs are reversed, have been found useful in this crossing minimization context.

2 Algorithm and Implementation

When the integrality conditions in (LO) are dropped, we obtain a linear programming relaxation of (LO) which has been proven very useful in practical applications. The structure of our branch&cut algorithm `oscm` (“one-sided crossing minimization”) is similar to the one proposed in [4]. The algorithm starts with the trivial constraints $0 \leq x_{ij} \leq 1$ that are

handled implicitly by the linear programming solver, iteratively adds violated cycle and Möbius-ladder constraints, and deletes nonbinding constraints after a linear program has been solved, until the relaxation is solved. This requires a *separation* algorithm that, given the solution of some relaxation, is able to determine a violated inequality called *cutting plane*. If the optimum solution of the relaxation is integral, the algorithm stops, otherwise it is applied recursively to two subproblems in one of which a fractional x_{ij} is set to 1 and in the other set to 0. Thus, in the end, an optimum solution is found as the solution of some relaxation, along with a proof of optimality.

`oscm` makes use of the following observations, some of which stem from the literature in fixed-parameter algorithms for one-sided crossing minimization. Lemma 1 allows us to decompose the given instance. Within the components, we can fix and eliminate variables from (LO) by Lemma 2, and we can exclude variables x_{ij} with $a_{ij} = 0$ from (LO) by Lemma 3.

► **Lemma 1 (Decomposition).** *For each node $v \in B$, we define the open interval $I_v =]l_v, r_v[$, where l_v is the position of the leftmost and r_v the position of the rightmost neighbor of v in π_T . The union of the intervals I_v induces a partition B_1, B_2, \dots, B_k of B such that every $I_{B_i} = \bigcup_{v \in B_i} I_v$, $i = 1, \dots, k$, is an interval, and for any pair B_i, B_j the intervals I_{B_i} and I_{B_j} are disjoint. In every optimum π_B all the nodes of B_i appear before those of B_j if I_{B_i} is to the left of I_{B_j} .*

Indeed, 51 of the 100 exact-public instances have between 2 and 154 components.

► **Lemma 2 (Variable fixing [1]).** *If for any pair of nodes $b_i, b_j \in B$, we have $c_{ij} = 0$ and $c_{ji} > 0$, then every optimal solution of (LO) satisfies $x_{ij} = 1$, if $i < j$, and $x_{ji} = 0$, if $i > j$.*

► **Lemma 3 (Arbitrary ordering).** *Let $\pi_B^{(p)}$ be a partial ordering induced by the variables x_{ij} with $a_{ij} \neq 0$, then there exist values $x_{ij} \in \{0, 1\}$ for $a_{ij} = 0$ defining a total ordering π_B of B with no effect to the objective function value. This assignment can be found by topologically sorting B with respect to $\pi_B^{(p)}$.*

This setup has the advantage that (sometimes considerably) smaller linear programs need to be solved, but, on the other hand, separation becomes more involved. In order to obtain an optimal partial ordering $\pi_B^{(p)}$ of B using the variables left in (LO), we need to include cycle constraints for larger cycles as already mentioned in Section 1.

For computational efficiency, `oscm` has a hierarchy of separation procedures. The first for 3-dicycles is based on depth first search. The second for dicycles of length at least 4 with integral weights is also based on depth first search. Violated dicycles are shortened via breadth first search, restricted to the cycle nodes, starting from back arcs of the preceding depth first search. The third applies shortest path techniques for separation of cycles containing fractional arcs as described for the related acyclic subdigraph problem in section 5 of [6]. First, the above separation procedures are applied on the graph containing only the arcs present in (LO). If all of the above do not find any violated inequalities, `oscm` extends the search to the fixed arcs. After separation, the linear program is resolved using the dual simplex method providing the same or a better lower bound on the minimum number of crossings. If the progress compared to the previous bound is small for a sequence of such lower bounds, `oscm` applies a heuristic for finding violated Möbius ladder inequalities, and if this does not lead to a significant improvement, the branch&cut phase is started.

Whenever a linear program has been solved, it is checked by topological sorting if the solution is the characteristic vector of a linear ordering. If not, a relaxed topological sorting procedure is applied in the pursuit of finding a better incumbent solution that provides an upper bound for the minimum number of crossings. `oscm` stops when the (integral) upper bound and the (possibly fractional) lower bound differ by less than 1, proving optimality.

For small instances, `oscm` applies a variant of the heuristic “Kernighan-Lin 2” of [12] for finding a decent initial solution before the optimization starts.

3 Performance

Our program `oscm`, published in [9] and [8], consists of roughly 3500 lines of C/C++ code. It makes use of the coin-or [11] `Cbc` library, version 2.10.7 [3].

We submitted `oscm` both to the exact track and the parameterized track of PACE 2024. In the official ranking, `oscm` received the first place in the exact track with 199 of the 200 instances instances solved in about 5682 seconds, and the third place in the parameterized track with all 200 instances solved in about 25 seconds.

References

- 1 Vida Dujmović and Sue Whitesides. An efficient fixed parameter tractable algorithm for 1-sided crossing minimization. *Algorithmica*, 40(1):15–31, 2004. doi:10.1007/S00453-004-1093-2.
- 2 Peter Eades and Nicholas C. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11(4):379–403, 1994. doi:10.1007/BF01187020.
- 3 John Forrest et al. coin-or/Cbc: Release 2.10.7, 2022. URL: <https://zenodo.org/records/5904374>.
- 4 Martin Grötschel, Michael Jünger, and Gerhard Reinelt. A cutting plane algorithm for the linear ordering problem. *Operations Research*, 32(6):1195–1220, 1984. doi:10.1287/opre.32.6.1195.
- 5 Martin Grötschel, Michael Jünger, and Gerhard Reinelt. Facets of the linear ordering polytope. *Mathematical Programming*, 33(1):43–60, 1985. doi:10.1007/BF01582010.
- 6 Martin Grötschel, Michael Jünger, and Gerhard Reinelt. On the acyclic subgraph polytope. *Mathematical Programming*, 33(1):28–42, 1985. doi:10.1007/BF01582009.
- 7 Patrick Healy and Nikola S. Nikolov. Hierarchical drawing algorithms. In Roberto Tamassia, editor, *Handbook of Graph Drawing and Visualization*, Discrete Mathematics and Its Applications, pages 409–453. CRC Press, 2013. URL: <https://api.semanticscholar.org/CorpusID:7736149>.
- 8 Michael Jünger, Paul J. Jünger, Petra Mutzel, and Gerhard Reinelt. PACE 2024 – MPPEG, June 2024. Software, version 1.2. (visited on 2024-11-28). doi:10.5281/zenodo.11546972.
- 9 Michael Jünger, Paul J. Jünger, Petra Mutzel, and Gerhard Reinelt. PACE2024, June 2024. Software, version 1.0. (visited on 2024-11-28). URL: <https://github.com/pauljngr/PACE2024>, doi:10.4230/artifacts.22523.
- 10 Michael Jünger and Petra Mutzel. 2-layer straightline crossing minimization: Performance of exact and heuristic algorithms. *Journal of Graph Algorithms and Applications*, 1(1):1–25, 1997. doi:10.7155/jgaa.00001.
- 11 R. Lougee-Heimer. The common optimization interface for operations research: Promoting open-source software in the operations research community. *IBM Journal of Research and Development*, 47(1):57–66, 2003. doi:10.1147/rd.471.0057.
- 12 Rafael Martí and Gerhard Reinelt. *Exact and Heuristic Methods in Combinatorial Optimization – A Study on the Linear Ordering and the Maximum Diversity Problem*. Applied Mathematical Sciences. Springer Berlin, Heidelberg, 2022. doi:10.1007/978-3-662-64877-3.
- 13 PACE 2024 Web Page. URL: <https://pacechallenge.org/2024>.

PACE Solver Description: UzL Exact Solver for One-Sided Crossing Minimization

Max Bannach  

European Space Agency, Noordwijk, The Netherlands

Florian Chudigiewitsch  

Institute for Theoretical Computer Science, University of Lübeck, Germany

Kim-Manuel Klein  

Institute for Theoretical Computer Science, University of Lübeck, Germany

Marcel Wienöbst¹  

Institute for Theoretical Computer Science, University of Lübeck, Germany

Abstract

This document contains a short description of our solver *pingpong* for the one-sided crossing minimization problem that we submitted to the exact and parameterized track of the PACE challenge 2024. The solver is based on the well-known reduction to the weighted directed feedback arc set problem. This problem is tackled by an implicit hitting set formulation using an integer linear programming solver. Adding hitting set constraints is done iteratively by computing heuristic solutions to the current formulation and finding cycles that are not yet “hit.” The procedure terminates if the exact hitting set solution covers all cycles. Thus, optimality of our solver is guaranteed.

2012 ACM Subject Classification Theory of computation → Integer programming; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases integer programming, exact algorithms, feedback arc set, crossing minimization

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.28

Supplementary Material *Software (Exact Track)*: <https://github.com/mwien/pingpong>

Software (Parameterized Track): <https://github.com/mwien/pingpong-light>

1 Introduction

One-sided crossing minimization is a fundamental problem in graph drawing. For a given bipartite graph $G = (V_1 \cup V_2, E)$ and a linear ordering τ of V_1 , the goal is to find a linear ordering π of V_2 that minimizes the number of *crossings*, i.e., tuples $(\{u_1, u_2\}, \{v_1, v_2\}) \in E^2$ such that $\tau^{-1}(u_1) < \tau^{-1}(v_1)$ and $\pi^{-1}(u_2) > \pi^{-1}(v_2)$, with $\tau^{-1}(x)$ and $\pi^{-1}(x)$ being the position of vertex x in the respective ordering. Vice versa $\pi(i)$ and $\tau(i)$ denote the vertex at position i in π and τ , respectively.

The objective can be formulated concisely using the notion of crossing numbers: If c_{uv} is the number of crossings of edges incident to u or v given that u is ordered to the left of v , the goal is to find a linear ordering π that minimizes

$$\sum_{i=1}^{|V_2|} \sum_{j=i+1}^{|V_2|} c_{\pi(i)\pi(j)}.$$

¹ Corresponding author



In any linear ordering, either u comes before v or the other way around and, hence, it suffices to consider the difference $c_{uv} - c_{vu}$. A sensible objective function is thus:

$$\min_{\pi} \sum_{i=1}^{|V_2|} \sum_{j=i+1}^{|V_2|} \max(c_{\pi(i)\pi(j)} - c_{\pi(j)\pi(i)}, 0)$$

This formulation is well-known to be identical to the weighted directed feedback arc set problem with arc $u \rightarrow v$ having weight $c_{vu} - c_{uv}$ if this weight is positive. To solve this instance, it is obvious that each strongly connected component can be considered separately. These insight already renders a considerable amount of instance trivially solvable.

2 Implicit Hitting Set Formulation of Feedback Arc Set

The *feedback arc set* problem (FAS) can be expressed as an instance of the *hitting set* problem: Add a set per cycle containing all its edges. Clearly, such a formulation in its explicit form is infeasible with the number of cycles growing exponentially in the size of the FAS instance.

A better approach is to start with a small set of cycles and iteratively add more constraints. This approach is known as the *implicit hitting set* algorithm [3] and, in the context of integer programming, also referred to as *lazy constraint generation* or *row generation*. In its simplest form (which can be refined in many ways), such an iterative procedure could look like the following for FAS:

1. Initialize the set of cycle constraints C in some way.
2. Repeatedly,
 - a. find the optimal solution of the hitting set instance C and
 - b. check if this solution is an FAS. If this is the case, then terminate and output the solution. If not, then add cycles to C that are not “hit” by the current solution.

The procedure terminates only when the optimal hitting set solution contains an edge per cycle in the FAS instance, guaranteeing correctness. Importantly, the algorithm often terminates before all cycles of G were added, thus making it more practical than the naive explicit formulation mentioned above.

An implementation of this general method for solving FAS has recently been described in [1] and was discussed earlier in the form of a branch-and-cut algorithm in [5]. In both cases, integer programming is used to find the optimal solution to the hitting set instance. Denoting the number of vertices in the FAS instance by n , the number of edges by m and having a variable x_i per edge with weight w_i as given in the FAS instance, one obtains

$$\begin{aligned} \min_x \quad & \sum_{i=1}^m w_i x_i \\ \text{s.t.} \quad & \sum_{x_i \in C_j} x_i \geq 1, \quad \text{for each } j = 1, 2, \dots, |C| \\ & x_i \in \{0, 1\} \quad \text{for all } i \in \{1, \dots, n\}, \end{aligned}$$

where the hitting set constraints are added lazily until there are no more violations.

We note that the same constraints could also be expressed with a MaxSAT formulation. However, early benchmarks showed that MaxSAT solvers perform significantly worse for the instances in this challenge obtained through the reduction from the one-sided crossing minimization. Hence, we abandoned this approach and focused on the ILP formulation.

3 Description of Our Approach

We follow the general method described above closely. In this section, we provide more details regarding our concrete implementation.

We solely used cycles of length three as the initial set of cycles C for most of the challenge and this approach is still used for larger instances (where a subset of all 3-cycles is randomly selected). Shortly before the submission deadline, we switched to a more involved generation procedure for the other instances, in which we first run our heuristic solver [2] to find a good initial feedback arc set and generate cycles based on it (see more details below on how new cycles are generated). The improvements here are, however, minor even in cases when the heuristic can already identify the optimal solution.²

Generally, cycles are added based on utilizing a (not necessarily optimal) solution of the current hitting set formulation. For this, we consider the subgraph obtained by removing all edges in this solution set. In the resulting subgraph, an FAS is found by a heuristic, which builds the topological order greedily from left to right,³ and for each edge $u \rightarrow v$ in this FAS, new cycles are generated by finding short paths from v to u using a breadth-first search. Our implementation of the approach from the previous section can be described as follows:

1. Initialize the set of cycle constraints C .
2. Find a hitting set based on a degree-based heuristic and add cycle constraints as described above. If a non-empty set of cycles is added, repeat this step.
3. Find a hitting set based on a rounded LP solution and add cycle constraints as described above. If a non-empty set of cycles is added, repeat this step.
4. If the objective value of the rounded LP solution and the fractional LP solution differ by less than one, the found hitting set is optimal. In this case, output the FAS and terminate.
5. Try to add violated cycle constraints based on the fractional LP solution. If a non-empty set of cycles is added, go to step 2.
6. Find a hitting set by solving the ILP optimally. If this set is not an FAS, add further constraints as described above and go to step 2. Else output the FAS and terminate.

So, instead of directly starting an exact solver on the given hitting set instance, we first use heuristics for the purpose of adding new cycle constraints. Due to this interplay between the heuristics for hitting set and the ones for FAS that are combined to generate cycle constraints, we name our solver *pingpong*.

As hitting set heuristics we use, on the one hand, a simple degree-based heuristic (which has the advantage of being extremely fast) and, on the other hand, a relaxed version of the LP without the integrality constraint enforced (i.e., having $0 \leq x_i \leq 1$). For cycle generation, we round the solution and further improve it iteratively by a simulated annealing scheme. Further constraints are added this way as long as the heuristic solution does not cover all cycles, i.e., is not a valid FAS. Our goal is to start the ILP solver only a few times – or not at all if the rounded LP solution matches the objective value of the fractional LP solution.

Once this subgraph is acyclic, we try to add further cycles based on the fractional LP solution. We do this only once as this procedure, which is based on Dijkstra's algorithm [4] to find violated LP constraints, has significantly larger computational cost compared to the

² Interestingly, our heuristic solver finds optimal solutions quickly even for some of the instances our exact solver fails to solve within the 30 minute time limit. This shows that proving the lower bound is the main problem for these inputs.

³ The next vertex to place in the ordering is chosen such that the sum of violated edge weights is minimized.

BFS cycle generation procedure. If it adds no further cycles, the integer program is started and solved optimally. In the same way as before, the solution of the integer program is used to add further cycles, unless it already hits every cycle – in this case we found the optimum and terminate.

We used the HiGHS ILP solver [6] without any further tuning. As this solver does not yet offer lazy constraint generation, we restarted the solver for any new hitting set instance. In the future, it would be interesting to analyze how much gains could be made when implementing a lazy constraint callback.

Our algorithm manages to solve 195 of the 200 total instances in the exact track. Due to randomness in the cycle generation procedure the run-time can fluctuate for some instances. For the parameterized track, we submitted a simplified version of our solver, which gives slightly better performance on smaller and easier instances. For example, it does not use the degree heuristic for hitting set and the initial cycles are simply all cycles of length 3. The solver does not make use of the given cutwidth ordering – still, it solves all instances in about a minute total.

References

- 1 Ali Baharev, Hermann Schichl, Arnold Neumaier, and Tobias Achterberg. An exact method for the minimum feedback arc set problem. *Journal of Experimental Algorithmics (JEA)*, 26:1–28, 2021.
- 2 Max Bannach, Florian Chudigiewitsch, Kim-Manuel Klein, Till Tantau, and Marcel Wienöbst. UzL heuristic solver for one-sided crossing minimization. Technical report, University of Lübeck, 2024.
- 3 Karthekeyan Chandrasekaran, Richard M. Karp, Erick Moreno-Centeno, and Santosh S. Vempala. Algorithms for implicit hitting set problems. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23–25, 2011*, pages 614–629, 2011. doi:10.1137/1.9781611973082.48.
- 4 Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959. doi:10.1007/BF01386390.
- 5 Martin Grötschel, Michael Jünger, and Gerhard Reinelt. A cutting plane algorithm for the linear ordering problem. *Operations research*, 32(6):1195–1220, 1984. doi:10.1287/OPRE.32.6.1195.
- 6 Qi Huangfu and JA Julian Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10(1):119–142, 2018. doi:10.1007/S12532-017-0130-5.

PACE Solver Description: CRGone

Alexander Dobler  

TU Wien, Austria

Abstract

We describe *CRGone*, our solver for the exact and parameterized track of the Pace Challenge 2024. It solves the problem of one-sided crossing minimization, is based on an integer linear programming (ILP) formulation with additional reduction rules, and is implemented in C++ using the ILP solver SCIP with *Soplex*.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Human-centered computing → Graph drawings; Mathematics of computing → Permutations and combinations

Keywords and phrases Pace Challenge 2024, One-Layer Crossing Minimization, Exact Algorithm

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.29

Supplementary Material

Software (Source Code): <https://zenodo.org/doi/10.5281/zenodo.11634869>

Funding *Alexander Dobler*: Supported by the Vienna Science and Technology Fund (WWTF) under grant 10.47379/ICT19035.

1 Introduction

One-sided crossing minimization (OSCM) is a problem from layered graph drawing and was first introduced by Sugiyama et al. [11]. The input is a bipartite graph $G = (V_t \dot{\cup} V_b, E)$, $E \subseteq V_t \times V_b$, with a fixed order π_t of V_t . The question is to find an ordering π_b of V_b , that minimizes the number of edge crossings when G is drawn straight-line such that V_t and V_b are drawn on two respective horizontal lines ℓ_t and ℓ_b ordered according to π_t and π_b , respectively. It is a purely combinatorial problem as two edges $(a, b), (c, d) \in V_t \times V_b$ cross if and only if

- $a \prec_{\pi_t} c$ and $d \prec_{\pi_b} b$, or
- $c \prec_{\pi_t} a$ and $b \prec_{\pi_b} d$

where $x \prec_{\pi} y$ means that x comes before y in the order π . The problem is NP-hard [4], heuristics [11, 4], and fixed-parameter algorithms with the natural parameter [3, 2, 8] are available. It has also been extensively studied with regard to integer linear programming [7].

In Section 2 we give some definitions and in Section 3 we describe our solver.

2 Definitions and Problem Insights

We assume that the input graph is a multigraph as some of our modifications introduce multiedges. For a set X , let $\binom{X}{i}$ be all the subsets of X of size i . For a vertex u , let $N(u)$ be its adjacent vertices, and let $E(u)$ be its incident edges. Given $u \in V_b$, we define $s(u)$ as the open interval (a, b) where a is the minimum index of a neighbour of u in π_t , and b is the maximum index. For $u, v \in V_b$ ($u \neq v$), let $c(u, v)$ be the number of crossings between edge pairs in the set $E(u) \times E(v)$ when u is placed before v in π_b . We have that $\sum_{\{u, v\} \in \binom{V_b}{2}} \min(c(u, v), c(v, u))$ is a lower bound for the number of crossings and $\sum_{\{u, v\} \in \binom{V_b}{2}} \max(c(u, v), c(v, u))$ is an upper bound. We define $c_r(u, v) = c(u, v) - \min(c(u, v), c(v, u))$.



© Alexander Dobler;

licensed under Creative Commons License CC-BY 4.0

19th International Symposium on Parameterized and Exact Computation (IPEC 2024).

Editors: Édouard Bonnet and Paweł Rzażewski; Article No. 29; pp. 29:1–29:4

Leibniz International Proceedings in Informatics



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

For an instance G of OSCM let G_d be the directed multi-graph with V_b as vertex set such that for $\{u, v\} \in \binom{V_b}{2}$, there are $c(u, v)$ arcs from v to u and there are $c(v, u)$ arcs from u to v . It is known that OSCM is equivalent to finding a minimum feedback arc set in G_d ; a topological order of G_d after removal of the minimum feedback arc set of size k corresponds to an order of π_b with k crossings.

3 Solver description

We describe now our solver. It starts applying several reduction rules and decomposition rules which split the instance into multiple smaller instances. Then an integer linear programming formulation is applied that was optimized for sparse graphs.

3.1 Reduction rules

Here, we describe reduction and decomposition rules and how they were implemented and applied. The first rule is applied during preprocessing and is due to twins in V_b , which can be contracted.

► **Reduction Rule 1.** *Let $X \subseteq V_b$ maximal such that $|X| > 1$ and $\forall u, v \in X : N(u) = N(v)$. Contract X into a single vertex that has multiedges of multiplicity $|X|$.*

We find such sets X using a trie. The values $c(u, v)$ for the reduced instance are only computed afterward. The next is due to the formulation as feedback arc set problem.

► **Decomposition Rule 2 ([9]).** *Let $T = (G_1, G_2, \dots, G_p)$ be a topological order of the strongly connected components of G_d . Then split the instance into G_1, G_2, \dots, G_p , whose individual solutions are then concatenated according to the topological order.*

We also implemented decomposition rules based on biconnected components of G_d [10], which almost never applied to the input instances, so it was not included in the final submission.

The next reduction rule fixes the relative order of pairs of vertices in V_b .

► **Reduction Rule 3 ([3]).** *If there exist $u, v \in V_b$ with $c(u, v) = 0$, fix $u \prec_{\pi_2} v$.*

The last reduction rule is more complicated and is related to modular decompositions of two-structures [6].

► **Decomposition Rule 4.** *Let $X \subsetneq V_b$, $|X| > 1$, such that*

$$\forall u, v \in X \forall w \in V_b \setminus X : c_r(u, w) = c_r(v, w) \wedge c_r(w, u) = c_r(w, v).$$

Then compute an optimal order π_1 of X for $G[V_t \cup X]$. Let G_c be the graph obtained from G by contracting X into a single vertex x . Compute an optimal order π_2 of $(V_b \setminus X) \cup \{x\}$ in the reduced instance G_c . By replacing in π_2 the contracted vertex x by π_1 , we obtain an optimal solution.

The sets X above not containing a randomly chosen $y \in V_b$ are computed using a partition refinement algorithm as described in [6]. The above decomposition rules are applied recursively with decreasing priority, i.e., Decomposition Rule 2 has the highest priority, Reduction Rule 3 is only applied afterward to the decomposed parts, Decomposition Rule 4 has the lowest priority. We also implemented the reduction rules, which fix relative orders of vertex-pairs based on 2/1-structures from [2] with the same priority as Reduction Rule 3.

3.2 Integer linear program

If no decomposition and reduction rules are applicable, we employ an integer linear program. The formulation is as in [7]. Assume a total order $<$ on V_b . For each pair $u, v \in V_b$, $u < v$ we have a binary *ordering variable* $x_{u,v}$ which is 1 if and only if $u \prec_{\pi_b} v$. The formulation is as follows.

$$\begin{aligned} \min \quad & \sum_{u,v \in V_b, u < v} (c(v, u) + x_{u,v}(c(u, v) - c(v, u))) && \text{(ILP)} \\ 0 \leq x_{u,v} + x_{v,w} - x_{u,w} \leq 1 \quad & u, v, w \in V_b, u < v < w && \text{(TRANS)} \\ x_{u,v} \in \{0, 1\} \quad & u, v \in V_b, u < v && \text{(BIN)} \end{aligned}$$

Adaptations. Due to Reduction Rule 3 and the reduction rules from [2] we know the relative order of specific pairs of vertices from V_b . This means that for some ordering variables $x_{u,v}$ we already know that they are 1 or 0. We remove those variables from the model and replace them by the corresponding constant in the above model. Resulting constraints which are satisfied regardless of variable assignment are removed.

Next, the (TRANS)-constraints are separated using a branch and cut approach. This involves first categorizing the (TRANS) constraints based on the values of $s(u), s(v), s(w)$ for the vertices u, v, w in each constraint.

- If $s(u) \cap s(v) \cap s(w) \neq \emptyset$, then it is a *type-1* constraint.
- If a constraint is type-1 and additionally, there are two pairs x, y and p, q among the triple u, v, w such that $c(x, y) \neq c(y, x)$ and $c(p, q) \neq c(q, p)$, then it is *weak-type-1*.
- If a constraint is not type-1, it is a *type-2* constraint.

The idea is that type-1 constraints can be enumerated quickly without storing them by using a sweep-line over the sorted interval borders of $s(u)$ for all $u \in V_b$. We enumerate type-2 constraints by saving for each vertex $u \in V_b$ the set $S(u)$ of vertices $v \in V_b$ with $s(u) \cap s(v) \neq \emptyset$. Then the type-2 constraints (which stay after applying reduction rules) can be found by enumerating pairs in $S(u)$ for each $u \in V_b$. The solver is initialized without any (TRANS)-constraints. First, only violated weak-type-1 constraints are separated. Once, there is a separation round where no violated weak-type-1 constraint can be separated, type-1 constraints are separated from now on. Lastly, type-2 constraints are only separated in a separation round if there are no type-1 constraints that can be separated.

Lastly, we implemented a heuristic that exploits fractional solutions. To this end, we start with a feasible solution $\hat{\pi}_b$ conforming to the reduction rules. Then we compute values $p(u)$ for all $u \in V_b$: for each ordering variable $x_{u,v}$ let $y_{u,v}$ be the rounded value in the fractional solution. We add 1 to $p(v)$ if $y_{u,v} = 1$, otherwise, we add 1 to $p(u)$. Lastly, for all $u \in V_b$ we add to $p(u)$ the number of $v \in V_b$ that are fixed before u according to the reduction rules. The heuristic computes an ordering starting with $\hat{\pi}_b$ and swapping two adjacent vertices u, v in $\hat{\pi}_b$ (u before v) where $p(u) > p(v)$ and u does not have to be before v according to the reduction rules. This is implemented in a bubble-sort-like algorithm. The worst-case runtime is equal to the number of ordering variable in the reduced model.

The above is implemented in C++17 using SCIP version 9 [1] with Soplex version 7 [5]. Parameters are chosen such that the solver remains in the root of the branch and bound tree as long as possible.



References

- 1 Suresh Bolusani, Mathieu Besançon, Ksenia Bestuzheva, Antonia Chmiela, João Dionísio, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Mohammed Ghannam, Ambros Gleixner, et al. The SCIP optimization suite 9.0, 2024. [arXiv:2402.17702](https://arxiv.org/abs/2402.17702).
- 2 Vida Dujmovic, Henning Fernau, and Michael Kaufmann. Fixed parameter algorithms for one-sided crossing minimization revisited. *J. Discrete Algorithms*, 6(2):313–323, 2008. doi:10.1016/J.JDA.2006.12.008.
- 3 Vida Dujmovic and Sue Whitesides. An efficient fixed parameter tractable algorithm for 1-sided crossing minimization. *Algorithmica*, 40(1):15–31, 2004. doi:10.1007/S00453-004-1093-2.
- 4 Peter Eades and Nicholas C. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11(4):379–403, 1994. doi:10.1007/BF01187020.
- 5 Gerald Gamrath, Daniel Anderson, Ksenia Bestuzheva, Wei-Kun Chen, Leon Eifler, Maxime Gasse, Patrick Gemander, Ambros Gleixner, Leona Gottwald, Katrin Halbig, et al. The SCIP optimization suite 7.0, 2020. URL: <https://opus4.kobv.de/opus4-zib/files/7802/scipopt-70.pdf>.
- 6 Michel Habib and Christophe Paul. A survey of the algorithmic aspects of modular decomposition. *Comput. Sci. Rev.*, 4(1):41–59, 2010. doi:10.1016/J.COSREV.2010.01.001.
- 7 Michael Jünger and Petra Mutzel. Exact and heuristic algorithms for 2-layer straightline crossing minimization. In Franz-Josef Brandenburg, editor, *Proc. Symposium on Graph Drawing and Network Visualizations (GD'95)*, volume 1027 of LNCS, pages 337–348. Springer, 1995. doi:10.1007/BFB0021817.
- 8 Yasuaki Kobayashi and Hisao Tamaki. A fast and simple subexponential fixed parameter algorithm for one-sided crossing minimization. *Algorithmica*, 72(3):778–790, 2015. doi:10.1007/S00453-014-9872-X.
- 9 Sungju Park and Sheldon B Akers. An efficient method for finding a minimal feedback arc set in directed graphs. In *Proc. International Symposium on Circuits and Systems (ISCAS'92)*, volume 4, pages 1863–1866. IEEE, 1992.
- 10 Hermann Stamm. On feedback problems in planar digraphs. In Rolf H. Möhring, editor, *Proc. Graph-Theoretic Concepts in Computer Science (WG'90)*, volume 484 of LNCS, pages 79–89. Springer, 1990. doi:10.1007/3-540-53832-1_33.
- 11 Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. Methods for visual understanding of hierarchical system structures. *IEEE Trans. Syst. Man Cybern.*, 11(2):109–125, 1981. doi:10.1109/TSMC.1981.4308636.

PACE Solver Description: Crossy – An Exact Solver for One-Sided Crossing Minimization

Tobias Röhr 

Hasso Plattner Institute, University of Potsdam, Germany

Kirill Simonov  

Hasso Plattner Institute, University of Potsdam, Germany

Abstract

We describe Crossy, an exact solver for One-sided Crossing Minimization (OSCM) that ranked 5th in the Parameterized Algorithms and Computational Experiments (PACE) Challenge 2024 (Exact and Parameterized Track). Crossy applies a series of reductions and subsequently transforms the input into an instance of Weighted Directed Feedback Arc Set (WDFAS), which is then formulated in incremental MaxSAT. We use the recently introduced concept of User Propagators for CDCL SAT solvers in order to dynamically add cycle constraints.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases One-sided Crossing Minimization, Exact Algorithms, Graph Drawing, Incremental MaxSAT

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.30

Supplementary Material *Software (Source Code)*: <https://github.com/roehrt/crossy>

archived at `swh:1:dir:aa5e0abc800f7a51008548897226b54b38032404`

Software (Source Code): <https://doi.org/10.5281/zenodo.12082773>

1 Preliminaries

Given a bipartite graph $G = (A, B, E)$ and a linear ordering on the vertices of A , the One-sided Crossing Minimization problem asks for a linear ordering \prec on the vertices of B that minimizes the number of crossings of a straight-line drawing when placing the vertices in A and B on two parallel lines in the respective order. To enable some of our reduction rules, it is convenient to relax the problem and allow the input to be a multigraph.

For $u, v \in B$, define $c(u, v)$ to be the number of crossings between the edges incident to u and v when $u \prec v$. Moreover, we call $u \prec v$ the *natural order* of u and v if and only if $c(u, v) < c(v, u)$. Since in any solution either $u \prec v$ or $v \prec u$, we get a simple lower bound on the number of crossings: $\sum_{u, v \in B} \min(c(u, v), c(v, u))$.

The penalty graph of an OSCM-instance is a directed graph on the vertices of B . In order to penalize pairs of vertices that do not appear in their natural order, we add an arc $u \rightarrow v$ carrying weight $c(u, v) - c(v, u)$ for any pair $u, v \in B$ with $c(u, v) > c(v, u)$. Note that the weight of a Minimum Weight Feedback Arc Set in the penalty graph equals the minimum number of crossings in the corresponding OSCM-instance above its lower bound [8].

We say that we *commit* $u \prec v$ if we only look for solutions where u appears before v . To model this knowledge in the penalty graph, we insert an arc $u \rightarrow v$ with infinite weight in this case.



© Tobias Röhr and Kirill Simonov;

licensed under Creative Commons License CC-BY 4.0

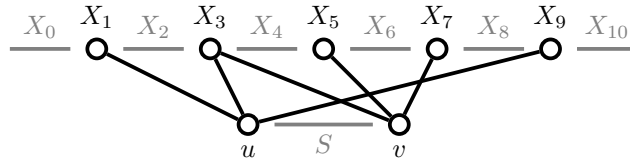
19th International Symposium on Parameterized and Exact Computation (IPEC 2024).

Editors: Édouard Bonnet and Paweł Rzażewski; Article No. 30; pp. 30:1–30:4

Leibniz International Proceedings in Informatics



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



■ **Figure 1** A configuration with $u \prec v$. S is the set of vertices between u and v . The sets X_i partition the neighborhood of S . The figure is adapted from [3].

2 Reduction rules

After merging twins in B and removing isolated vertices, we apply two sets of reduction rules: one specific to OSCM and the other consisting of general-purpose rules for the Weighted Directed Feedback Arc Set problem.

2.1 Rules for OSCM

We utilize two well-known rules for OSCM that we call Planar Ordering and Transitivity, and introduce a new rule, Dominance.

Planar Ordering If there is a pair of vertices $u, v \in B$ such that $c(u, v) = 0$, commit $u \prec v$.

Transitivity If $u \prec v$ and $v \prec w$, commit $u \prec w$.

The Planar Ordering reduction rule is due to Dujmovic and Whitesides [4]; see their work for the proof of correctness. The Transitivity rule is straightforward.

Before we formally define Dominance, consider the following argument showing that the natural order $u \prec v$ is optimal, in a certain case. Assume by contradiction that $u \prec v$ is not optimal, and consider an optimal ordering where v appears before u instead. Let S be the set of vertices between v and u in this ordering, i.e., $S = \{w \in B \mid v \prec w \prec u\}$.

In order for $u \prec v$ not to be optimal, the number of crossings caused by $v \prec S \prec u$ must be strictly less than the number of crossings caused by all configurations with $u \prec v$, namely: $u \prec S \prec v$, $u \prec v \prec S$, and $S \prec u \prec v$. If we can derive a contradiction for each possible set S , we have proven that $u \prec v$ is optimal and are able to commit $u \prec v$.

Inspired by the tabular analysis technique of Dujmovic, Fernau and Kaufmann [3], let us categorize the neighbors of S into disjoint sets based on their relative position to the neighbors of u and v , so that the neighbors in the same set are effectively indistinguishable with respect to the condition above (see Figure 1).

By introducing variables describing the cardinality of those sets, we can express the number of crossings for each configuration as a linear combination of these variables. Let $L(x, y, z)$ denote this linear combination for some configuration $x \prec y \prec z$. Now we can derive a system of linear inequalities that must hold for $u \prec v$ not to be optimal:

$$L(v, S, u) < L(S, u, v)$$

$$L(v, S, u) < L(u, S, v)$$

$$L(v, S, u) < L(u, v, S).$$

Additionally, each variable can also be bounded by the number of edges outgoing from each partition. Finally, we can use an LP solver to check the feasibility of this system of inequalities, leading us to the following rule:

Dominance If there is a pair of vertices $u, v \in B$ such that the LP derived from the above analysis is infeasible, commit $u \prec v$.

Thus, the Planar Ordering rule is a special case of the Dominance rule.

Although Dominance runs in polynomial time, it is computationally expensive as it requires solving a linear program for each vertex pair. To mitigate this, we use a weaker rule that removes upper bound constraints and checks only pairwise inequality feasibility.

This can be done in linear time, resulting in $\mathcal{O}(|B| \cdot |E|)$ for all OSCM-reductions.

2.2 Rules for WDFAS

Crossy proceeds to apply very general rules for the Weighted Directed Feedback Arc Set problem on the penalty graph.

Strongly Connected Components Find a solution for each strongly connected component, then combine the orderings following a topological ordering of the condensation graph.

Minimum Cut For each arc $u \rightarrow v$, commit $u \prec v$, if the weighted minimum cut separating v from u does not exceed the weight of $u \rightarrow v$.

Although the Minimum Cut rule can commit some pairs, our experiments show that the additional computational cost is not justified. We therefore disable it in our implementation, resulting in the overall running time of $\mathcal{O}(|B| \cdot |E|)$ for all preprocessing steps.

3 Incremental MaxSAT formulation

Building on the work of the winning team of the PACE Challenge 2022 [6], we formulate the Weighted Directed Feedback Arc Set problem as incremental MaxSAT problem, aiming to hit all cycles in the penalty graph.

3.1 Encoding

For each arc $u \rightarrow v$ in the penalty graph, we introduce a variable $x_{u \rightarrow v}$ representing the arc's inclusion in the solution. Each variable is assigned the negated weight of its corresponding arc. If an arc has infinite weight, the variable is set to false. To encode a cycle constraint, we add a disjunction of the variables corresponding to the arcs in that cycle.

Crossy starts by adding short cycles of length at most 4 to the MaxSAT instance explicitly. All longer cycles are added dynamically later by the user propagator.

3.2 User Propagator

We modify UWMaxSAT [7] to allow us to connect a user propagator to its underlying SAT solver, CaDiCal [1]. The recently introduced IPASIR-UP interface [5] enables us to connect user-defined propagators to CaDiCal without modifying the solver itself.

Our user propagator employs the concept of Cycle Propagation as introduced by Kiesel and Schidler [6]. Assigning a negative literal $x_{u \rightarrow v}$ indicates that the arc $u \rightarrow v$ is not included in the solution and, therefore, remains in the graph. Following the SAT solver's decision, our user propagator detects if the current assignment would close a cycle. If a cycle is detected, we add the corresponding cycle constraint to our MaxSAT instance, effectively pruning the current branch.

We tackle incremental cycle detection with rollbacks by eagerly maintaining the depth of each vertex. Upon each arc insertion, we recursively update the depths of affected vertices and check for any newly formed cycles.

As many arcs have infinite weight and thus always remain in the graph, we first compute the transitive reduction of this subgraph to reduce the workload for our cycle detection.

4 Discussion

While its performance secured Crossy the 5th place in PACE 2024, there are some challenges that arise due to its MaxSAT-based cycle-hitting formulation.

The penalty graph of an OSCM instance is known to be very dense, and our cycle-hitting formulation can result in a large, potentially exponential, number of constraints, even with the use of a user propagator. An alternative approach, employed by the top 3 teams, uses transitivity constraints for each triplet of vertices in B , which appears to perform better.

Moreover, OSCM allows for the application of various effective primal heuristics that can enhance ILP-based approaches but are not applicable to MaxSAT-based solvers like Crossy.

Finally, the performance of UWMaxSAT in MaxSAT competitions benefits from running SCIP [2] beforehand. This dependency poses a significant drawback for Crossy, as SCIP cannot take advantage of the user propagator.


References

- 1 Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froyleys, and Florian Pollitt. *Radical 2.0*. In Arie Gurfinkel and Vijay Ganesh, editors, *Computer Aided Verification – 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24–27, 2024, Proceedings, Part I*, volume 14681 of *Lecture Notes in Computer Science*, pages 133–152. Springer, 2024. doi:10.1007/978-3-031-65627-9_7.
- 2 Suresh Bolusani, Mathieu Besançon, Ksenia Bestuzheva, Antonia Chmiela, João Dionísio, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Mohammed Ghannam, Ambros Gleixner, et al. The scip optimization suite 9.0. *arXiv preprint*, 2024. arXiv:2402.17702.
- 3 Vida Dujmovic, Henning Fernau, and Michael Kaufmann. Fixed parameter algorithms for one-sided crossing minimization revisited. *J. Discrete Algorithms*, 6(2):313–323, 2008. doi:10.1016/J.JDA.2006.12.008.
- 4 Vida Dujmovic and Sue Whitesides. An efficient fixed parameter tractable algorithm for 1-sided crossing minimization. *Algorithmica*, 40(1):15–31, 2004. doi:10.1007/S00453-004-1093-2.
- 5 Katalin Fazekas, Aina Niemetz, Mathias Preiner, Markus Kirchweger, Stefan Szeider, and Armin Biere. IPASIR-UP: user propagators for CDCL. In Meena Mahajan and Friedrich Slivovsky, editors, *26th International Conference on Theory and Applications of Satisfiability Testing, SAT 2023, July 4–8, 2023, Alghero, Italy*, volume 271 of *LIPICs*, pages 8:1–8:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.SAT.2023.8.
- 6 Rafael Kiesel and André Schidler. PACE solver description: DAGer – Cutting out cycles with maxsat. In Holger Dell and Jesper Nederlof, editors, *17th International Symposium on Parameterized and Exact Computation, IPEC 2022, September 7–9, 2022, Potsdam, Germany*, volume 249 of *LIPICs*, pages 32:1–32:4. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.IPEC.2022.32.
- 7 Marek Piotrów. UWMaxSat: Efficient solver for maxsat and pseudo-boolean problems. In *32nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2020, Baltimore, MD, USA, November 9–11, 2020*, pages 132–136. IEEE, 2020. doi:10.1109/ICTAI50040.2020.00031.
- 8 Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiro Toda. Methods for visual understanding of hierarchical system structures. *IEEE Trans. Syst. Man Cybern.*, 11(2):109–125, 1981. doi:10.1109/TSMC.1981.4308636.



PACE Solver Description: CIMAT_Team

Carlos Segura  



Area de Computación, Centro de Investigación en Matemáticas (CIMAT), Guanajuato, Mexico

Lázaro Lugo  

Area de Computación, Centro de Investigación en Matemáticas (CIMAT), Guanajuato, Mexico

Gara Miranda  

Departamento de Ingeniería Informática y de Sistemas, Universidad de La Laguna, Spain

Edison David Serrano Cárdenas  

Area de Matemáticas Aplicadas, Centro de Investigación en Matemáticas (CIMAT), Guanajuato, Mexico

Abstract

This document describes MAEDM-OCM, a first generation memetic algorithm for the one-sided crossing minimization problem (OCM), which obtained the first position at the heuristic track of the Parameterized Algorithms and Computational Experiments Challenge 2024. In this variant of OCM, given a bipartite graph with vertices $V = A \cup B$, only the nodes of the layer B can be moved. The main features of MAEDM-OCM are the following: the diversity is managed explicitly through the Best-Non-Penalized (BNP) survivor strategy, the intensification is based on Iterated Local Search (ILS), and the cycle crossover is applied. Regarding the intensification step, the neighborhood is based on shifts and only a subset of the neighbors in the local search are explored. The use of the BNP replacement was key to attain a robust optimizer. It was also important to incorporate low-level optimizations to efficiently calculate the number of crossings and to reduce the requirements of memory. In the case of the longest instances ($|B| > 17000$) the memetic approach is not applicable with the time constraints established in the challenge. In such cases, ILS is applied. The optimizer is not always applied to the original graph. In particular, twin nodes in B are grouped in a single node.

2012 ACM Subject Classification Computing methodologies → Discrete space search

Keywords and phrases Memetic Algorithms, Diversity Management, One-sided Crossing Minimization

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.31

Supplementary Material *Software*: <https://github.com/carlossegurag/PaceChallenge24>
archived at `swh:1:dir:c7ae637013531bdb241615553d8bbcb6b25ab469`

Software: <https://zenodo.org/doi/10.5281/zenodo.12512615>

Acknowledgements We would like to thank to the High-Performance Computing (HPC) team of CIMAT and to CONAHCYT for the funding of the HPC laboratories.

1 Preliminaries

The One-sided Crossing Minimization problem (OCM) involves arranging the nodes of a bipartite graph in two layers, so that the crossing of edges is minimized when a straight-line drawing is performed. In the Parameterized Algorithms and Computational Experiments (PACE) Challenge, the vertices associated to each of the layers (A and B) as well as the order of the vertices in A are given. Thus, the problem seeks an order of B so as to minimize the number of crossings. Since any order of the nodes in B is valid, a natural encoding is a permutation of the vertices in B , which has been the encoding selected for our method.



© Carlos Segura, Lázaro Lugo, Gara Miranda, and Edison David Serrano Cárdenas; licensed under Creative Commons License CC-BY 4.0

19th International Symposium on Parameterized and Exact Computation (IPEC 2024).

Editors: Édouard Bonnet and Paweł Rzażewski; Article No. 31; pp. 31:1–31:4

Leibniz International Proceedings in Informatics



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

Algorithm 1 Memetic Algorithm with Explicit Diversity Management for OCM.

Require: $InitFactor$, N (size of population), *Stopping criterion Phase 1 (time1)*, *Global Stopping criterion (time2)*

- 1: **Initialization:** Generate an initial population P_0 with N individuals. Assign $i = 0$.
 - 2: **Iterated Local Search:** Apply Iterated Local Search to every individual in P_0 .
 - 3: **Diversity Initialization:** Calculate the initial desired minimum distance (D_0) as the mean distance among individuals in P_0 multiplied by $InitFactor$.
 - 4: **while** the execution time of Phase 1 (time1) has not been reached **do**
 - 5: **Mating Selection:** Perform binary tournament selection on P_i in order to fill the mating pool with N parents.
 - 6: **Variation:** Apply the cycle-based crossover (CX) in the mating pool to create the set O_i with N offspring.
 - 7: **Iterated Local Search:** Apply Iterated Local Search to every individual in O_i .
 - 8: **Survivor Selection:** Apply Best-Non Penalized survivor selection strategy (BNP) to create P_{i+1} by considering P_i and O_i as input.
 - 9: $i = i + 1$
 - 10: **end while**
 - 11: **Iterated Local Search:** Apply Iterated Local Search to the best evaluated permutation so far, until time2 is exhausted
 - 12: **Return** best evaluated permutation.
-

Regarding the solving strategies, Memetic Algorithms (MAs) are one of the most effective solvers for NP-hard problems. In fact, in several problems where solutions are encoded as permutations, MAs are the leading methods. This is the case of the Job-Shop Scheduling Problem [1] and the Linear Ordering Problem [3], among others. In these cases, the explicit management of diversity was key to develop robust methods that are able to reach high-quality solutions with a high probability. Thus, our team decided to adapt some of the principles that were successful in those problems to the OCM.

2 MAEDM-OCM: a first generation memetic algorithm for the one-sided crossing minimization problem

The Memetic Algorithm with Explicit Diversity Management for the One-sided Crossing Minimization problem (MAEDM-OCM) is a first-generation MA. MAEDM-OCM applies a set of operators that have already proven to be effective for permutation encoding. Given that short-term executions are performed, a first-generation MA is applied. Thus, a population-based approach is combined with a non-adaptive intensification scheme which in this case is Iterated Local Search (ILS). Algorithm 1 shows the general working operation of our proposal. Differently to most MAs, the method is divided in two phases. In the first phase (lines 1-10) a traditional MA is considered. In the second phase (line 11), ILS is applied to the best solution found so far. The reason to incorporate this second phase is that for medium and large instances, the stopping criterion used in the challenge is not enough to evolve a large number of generations. Thus, there might be opportunity for further improvements by applying ILS. In spite of this additional change, the most important decisions that affect the performance of MAEDM-OCM are related to the specific components that were used in the first phase. In the following, the working operation of each component of the first phase is described.

Our approach starts by initializing a population with N individuals, where each permutation is equiprobable (Line 1). Then, each solution is improved with ILS (Line 2). ILS uses a first-improvement stochastic hill-climber that considers the shift neighborhood [2]. This neighborhood is selected because it can be explored efficiently by storing the crossing number matrix. Neighbors are generated by moving a vertex in the given permutation to an alternative position and shifting all the vertices in the intermediate positions. However, not all the moves are taken into account. Each number is moved to the left and right until its worsening is larger than a threshold value or until it reaches the first or last position. Then, the best move is accepted. The worsening threshold is equal to the median value of the crossing number matrix multiplied by *cuttingMult*, which is a parameter of the optimizer. Regarding the perturbations performed by ILS, three different alternatives were used equiprobably: swap a set of *SwapSize* pairs of positions, do a random shuffle of a block with size *permBlock* or move a block with a size that is selected randomly between 1 and 10 to a random position of the permutation.

An important feature of our approach is that it considers diversity explicitly. This is done with the replacement strategy, which works by setting a minimum desired distance that is updated during the run. Similarly to [1], the distances that appear in the initial population are used to set the initial desired distance (D_0) (Line 3). In particular, D_0 is calculated as the mean distance among all the individuals in the initial population multiplied by *InitFactor*, which is a parameter of MAEDM-OCM. In order to calculate D_0 , the Spearman's footrule distance [4] is employed.

MAEDM-OCM evolves a set of generations until a given stopping criterion is reached (Lines 4-10). At each generation, a set of N parents is selected using binary tournaments (Line 5). Then, the cycle-based crossover is applied (Line 6), and ILS is used to intensify (Line 7). Finally, the diversity-aware replacement strategy called BNP is applied (Line 8). BNP is an elitist survivor selection strategy that avoids the survival of too close solutions. The meaning of too close is defined dynamically. At the initial stages it forces larger distances between solutions with the aim of promoting exploration, whereas at the final stages closer solutions are accepted with the aim of promoting exploitation. The details are given in [1].

2.1 Other Improvements and Treatment of Large Instances

There were several low-level optimizations that were important to the efficiency:

- The performance of the local search was improved by storing the crossing number matrix and an improvement matrix that contains the gain of swapping two consecutive nodes of a solution.
- The crossing number matrix is calculated efficiently by using data structures such as balanced search trees or the two-pointer technique, depending on the size of the instance.
- The data types for storing the matrices is adapted depending on the requirements of the instance.
- Twin vertices in B are grouped for creating a shorter graph with parallel edges that can be used to solve the original problem with a reduced search space.

In spite of the efforts for efficiency, the stopping time established for the challenge was not large-enough for using the two phases of MAEDM-OCM in the longest instances. In instances with $|B| > 17000$, only the second phase is used, i.e. it directly applies ILS. Moreover, in this case the solution is not created randomly. Instead, for each edge (a_i, b_i) an score is assigned which is equal to the amount of existing edges with its A -endpoint lower than a_i and its B -endpoint different to b_i . Then, each vertex of B is assigned an score equal to the mean of its adjacent edges. The initial solution greedily sorts the vertices in B by increasing score.

References

- 1 Oscar Hernández Constantino and Carlos Segura. A parallel memetic algorithm with explicit management of diversity for the job shop scheduling problem. *Appl Intell*, pages 1–13, April 2021.
- 2 Manuel Laguna, Rafael Martí, and Vicente Campos. Intensification and diversification with elite tabu search solutions for the linear ordering problem. *Comput Oper Res*, 26(12):1217–1230, 1999.
- 3 Lázaro Lugo, Carlos Segura, and Gara Miranda. A diversity-aware memetic algorithm for the linear ordering problem. *Memetic Computing*, 14(4):395–409, 2022. doi:10.1007/s12293-022-00378-5.
- 4 Marc Sevaux, Kenneth Sörensen, et al. Permutation distance measures for memetic algorithms with population management. In *Proceedings of 6th Metaheuristics International Conference, MIC'05*, pages 832–838, 2005.

PACE Solver Description: Martin_J_Geiger

Martin Josef Geiger  

University of the Federal Armed Forces Hamburg, Germany

Abstract

This extended abstract outlines our contribution to the Parameterized Algorithms and Computational Experiments Challenge (PACE), which invited to work on the one-sided crossing minimization problem. Our ideas are primarily based on the principles of Iterated Local Search and Variable Neighborhood Search. For obvious reasons, the initial alternative stems from the barycenter heuristic. This first sequence (permutation) of nodes is then quickly altered/ improved by a set of operators, keeping the elite configuration while allowing for worsening moves and hence, escaping local optima.

2012 ACM Subject Classification Mathematics of computing → Optimization with randomized search heuristics

Keywords and phrases PACE 2024, one-sided crossing minimization, Variable Neighborhood Search, Iterated Local Search

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.32

Supplementary Material *Software*: <https://doi.org/10.5281/zenodo.11465516>

Acknowledgements We would like to thank the organizers of PACE 2024, namely Philipp Kindermann, Fabian Klute, and Soeren Terziadis, for working really hard for this interesting competition. Besides, our thanks go to the sponsors NETWORKS (<https://www.thenetworkcenter.nl/>) and the wonderful platform <https://optil.io>.

1 Problem description and some reflections

1.1 The problem

In the one-sided crossing minimization problem, a graph $G = (V, E)$ is given, which consists of a vertex set V and an edge set E . G is bipartite as there is a partition of V into two disjoint subsets V_1, V_2 (hence, $V = V_1 \cup V_2$, $V_1 \cap V_2 = \emptyset$, and $E \subseteq V_1 \times V_2$). We now assume that the nodes of V_1 are arranged in a linear order and placed in one layer, while the ones of V_2 appear in another layer parallel to the first one. Therefore, edges between V_1 and V_2 may cross, depending on the sequence of nodes in V_1, V_2 . In its *one-sided* variation, the crossing minimization problem lies in arranging (ordering) the nodes in V_2 – while assuming a fixed linear order $<_1$ of V_1 – such that the total number of edge crossings is minimal.

Several applications for this problem can be found in the literature, with graph drawing as a prominent example [3].

1.2 A lower bound and a corollary

It follows that the solution to the problem can be characterized as finding a (cost-minimal) linear order $<_2$ for V_2 . In any such order, two nodes $a, b \in V_2$ can appear either ordered $a < b$ or $b < a$, and the crossings count c_{ab} or (XOR) c_{ba} are part of the optimal value. A trivial lower bound is obtained by considering all distinct pairs $a, b \in V_2$, and computing the sum over all $\min\{c_{ab}, c_{ba}\}$ -values.

Concept 1. Based on this lower bound computation, we can construct a digraph on V_2 , introducing arcs (a, b) iff $c_{ab} < c_{ba}$, and arcs (b, a) iff $c_{ba} < c_{ab}$. In some ideal cases, this digraph is acyclical, and an optimal ordering $<_2$ is quickly computed based on this



© Martin Josef Geiger;

licensed under Creative Commons License CC-BY 4.0

19th International Symposium on Parameterized and Exact Computation (IPEC 2024).

Editors: Édouard Bonnet and Paweł Rzażewski; Article No. 32; pp. 32:1–32:4

Leibniz International Proceedings in Informatics



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

preliminary input. Unfortunately, acyclicity is not always present. It follows that, in those cases, any linear order $<_2$ breaks at least one (often: some, several) cycles, and the problem can be reformulated as finding a minimal-cost cycle-breaking of the constructed digraph. Part of the process now becomes identifying the elementary circuits of the digraph, e. g. by means of [4], and breaking them in an optimal manner. In our experience, if G becomes “large”, this process becomes computationally difficult.

Concept 2. Alternative approaches directly construct and manipulate the linear order $<_2$ by considering a permutation of the nodes in V_2 , and hence *implicitly* break existing cycles by forcing transitivity over all binary relations of $a, b \in V_2$. In such a permutation, the order of nodes in V_2 is considered from “left” to “right”, identical to the order of the node-IDs in the permutation.

As the transition from the digraph into the permutation is a mapping from a higher into a lower dimensional space (i. e., a lossy compression), such approaches are more direct but fail to enumerate the cycles in a structured manner.

2 Submitted algorithm

Our approach is primarily based on the principles of Variable Neighborhood Search [2] and Iterated Local Search [5]. In the spirit of the classification above, we follow Concept 2, and consider permutations of nodes of V_2 .

The basic idea of Iterated Local Search is as follows. First, and starting from an initial solution, we run into a local optimum (and in our case, we apply Variable Neighborhood Search, i. e., a set of different neighborhoods, to find a best-possible one). Second, a “perturbation”-/ “shaking”-move is applied, trying to escape the local optimum. Finally, search continues from here, and the process is repeated until a termination criterion is reached (which is in most cases – and here – a maximum running time).

Obviously, keeping an elite solution makes sense and is incorporated in our concept, also.

2.1 Preprocessing and reductions

Reducing the size of the instance is beneficial. First, we exclude isolated nodes in V_2 , i. e., nodes that have no edges. Then, and excluding the very large instances, all c_{ab} -values are pre-computed. On this basis, the reduction rules RR1 and RR2, as given in [1], are applied. Applying those reduction rules is beneficial for most data sets, as we find partial orders on the set V_2 , and therefore can tell whether some nodes must precede others in the optimal solution.

If possible, V_2 is further broken down into linearly ordered, disjoint subsets, such that the nodes of each subset must precede the ones of the following subset in the permutation, etc. Each subset can then be treated as an independent sub-problem, and the search process is consequently accelerated. The main idea is to iterate through the nodes of the ordered set V_1 , starting with the first (leftmost) node and checking the incident edges for overlaps with edges of succeeding nodes (on the right of the currently considered one). This partitioning can be computed in $\mathcal{O}(|V_1| + |E|)$, and is therefore feasible in cases in which pre-computing the crossings-matrix is computationally too expensive.

2.2 Initial permutation of V_2

The starting solution stems from the barycenter-heuristic [6]. In this approach, the average positions of adjacent nodes of each node in V_2 are computed, and the nodes are subsequently sorted in non-decreasing order of those values.

Starting with this heuristics is important, as the challenge organizers have published some instances for which this approach yields the optimal solution. In those cases, our program terminates early. In the Heuristics-Track of the competition, this applies to 12 of the 100 instances.

2.3 Intensification: Improvement moves

We exhaustively search for improving moves until a local optimum is reached. The following neighborhoods are employed.

- First and foremost, the *single node move* tries to remove a node from its current position and re-insert in some other place in the permutation. Re-insertion positions are considered to the “left” and to the “right”, starting with close reinsertions and working the way up to ones further away.
- Besides, *block moves* try to move entire blocks of subsequent nodes. The size of the blocks range from 2 to 5 nodes. Our experiments indicate that block moves contribute to the performance of the algorithm only a little – but still they do.

Improving moves are always accepted, and moves that do not change the quality of the current solution are considered with a certain probability in order to diversify the search.

2.4 Speed-ups

Several truncation-techniques are implemented in order to speed-up the search. Obviously, moves that contradict the order given by the reduction rules RR1 and RR2 are omitted. Also, when moving a node (or a block), movements are stopped once their cumulative change in the objective function value exceeds a certain threshold: In those cases, we do not hope for an improvement to show up when trying to move the node even further.

For the larger instances, i. e. the ones in which computing the crossings matrix is considered to be computationally too expensive, we truncate the movements further by introducing a maximum range (change of positions) for shifting nodes in the permutation. This is important as the algorithm otherwise spends too much time re-inserting a given node before moving on to the next node.

2.5 Diversification: Perturbation move

Once a local optimum is reached, a subset of the permutation is reversed and search continues from here. We allow for a maximum of 20% of the permutation to be reversed. Based on our experiments, this value presents a good compromise between diversifying and intensifying the search.

2.6 Parameter tuning

Finding good values to the guiding parameters is not to be neglected. In fact, more than 24,000 runs have been conducted to find an appropriate setting, i. e., a single, identical setting for all test instances.

In this process, it has been verified that reversing up to 20% of the nodes the permutation in the diversification move is a good choice. Bigger percentages work, too, but diversify the search more and hence require relatively more computing time in order to find excellent results.

Also, nodes and blocks of nodes in V_2 are shifted at most ± 3900 positions to either “left” or “right”. Again, other values work also, but this number appears to be around the sweet-spot for the available instances.

3 Overall ranking and some insights

Despite being a relatively comprehensive approach, our concept ranked third in this very competitive challenge (the Heuristics-Track of PACE 2024). We attribute this to the rather well-done implementation, along with some clever speed-up techniques as outlined above.

Even more successful approaches (Memetic Algorithms) make use of an archive of local optima. Such concepts have not been used by us here, but given the direct comparison to the top-ranked team, such a algorithmic design element would have been beneficial, indeed.

4 Source-code

The source-code of our contribution has been published under the Creative Commons Attribution 4.0 International Public License and made available under <https://doi.org/10.5281/zenodo.11465516>.

References

- 1 Vida Dujmović, Henning Fernau, and Michael Kaufmann. Fixed parameter algorithms for ONE-SIDED CROSSING MINIMIZATION revisited. *Journal of Discrete Algorithms*, 6:313–323, 2008.
- 2 Pierre Hansen and Nenad Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130:449–467, 2001. doi:10.1016/S0377-2217(00)00100-4.
- 3 Patrick Healy and Nikola S. Nikolov. Hierarchical drawing algorithm. In Roberto Tamassia, editor, *Handbook of Graph Drawing and Visualization*, Discrete Mathematics and its Applications, chapter 13, pages 409–453. CRC Press – Taylor Francis Group, Boca Raton, FL, USA, June 2013.
- 4 Donald B. Johnson. Finding all elementary circuits in a directed graph. *SIAM Journal on Computing*, 4(1):77–84, 1975.
- 5 Helena R. Lourenço, Olivier Martin, and Thomas Stützle. Iterated local search. In Fred Glover and Gary A. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, chapter 11, pages 321–353. Kluwer Academic Publishers, Boston, Dordrecht, London, 2003. doi:10.1007/0-306-48056-5_11.
- 6 Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11(2):109–125, February 1981. doi:10.1109/TSMC.1981.4308636.

PACE Solver Description: Arcee

Kimon Boehmer ✉

Université Paris-Saclay, France

Lukas Lee George ✉ 

Technical University Berlin, Germany

Fanny Hauser ✉

Technical University Berlin, Germany

Jesse Palarus ✉ 

Technical University Berlin, Germany

Abstract

The 2024 PACE Challenge focused on the ONE-SIDED CROSSING MINIMIZATION (OCM) problem, which aims to minimize edge crossings in a bipartite graph with a fixed order in one partition and a free order in the other. We describe our OCM solver submission that utilizes various reduction rules for OCM and, for the heuristic track, employs local search approaches as well as techniques to escape local minima. The exact solver uses an ILP formulation and branch & bound to solve an equivalent FEEDBACK ARC SET instance.

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

Keywords and phrases PACE 2024, One-Sided Crossing Minimization, OCM

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.33

Supplementary Material

Software (Source Code): https://github.com/lucidLuckylee/pace_2024 [2]

archived at `swh:1:dir:4a1810b816bec954e0780d591b0c79276ac8f285`

Acknowledgements We want to thank Mathias Weller and André Nichterlein for their advice and the *Internet Architecture and Management Research Group TU Berlin* for their hardware support.

1 Introduction

In the Parameterized Algorithms and Computational Experiments (PACE) Challenge of 2024, the problem of interest was ONE-SIDED CROSSING MINIMIZATION (OCM). In this problem, we are given a bipartite graph with vertex partitions A and B which are drawn horizontally and in parallel. A comes with a fixed linear order and is thus called the set of *fixed* vertices, while the ordering of the vertices in B is unknown (we call these vertices *free*). The goal is to find an ordering of the vertices in B that minimizes the total number of edge crossings when all edges are drawn with straight lines. Usually, exact and heuristic solvers for OCM will first require the computation of the so-called *crossing matrix* or *crossing numbers* [4, 6, 7, 11]. An entry c_{uv} with $u, v \in B$ denotes the number of edge crossings between edges incident to u and edges incident to v , when u appears before v in the ordering. The *penalty graph* of our OCM instance is a directed graph (B, E) where $E := \{(u, v) \in B^2 \mid c_{uv} < c_{vu}\}$ with edge weights $w((u, v)) = c_{vu} - c_{uv}$. Sugiyama et al. [15] observed a connection between OCM and the Feedback Arc Set of the penalty graph: An optimal ordering of the vertices in B for OCM is equal to a topological ordering when an optimal Feedback Arc Set is removed from the penalty graph. In the following, we consider OCM instances and graphs to be *large* if the solver opts not to generate and store their crossing matrix and penalty graph due to memory limitations. In our submitted solver all instances with more than 10,000 free vertices are considered large.



© Kimon Boehmer, Lukas Lee George, Fanny Hauser, and Jesse Palarus;
licensed under Creative Commons License CC-BY 4.0

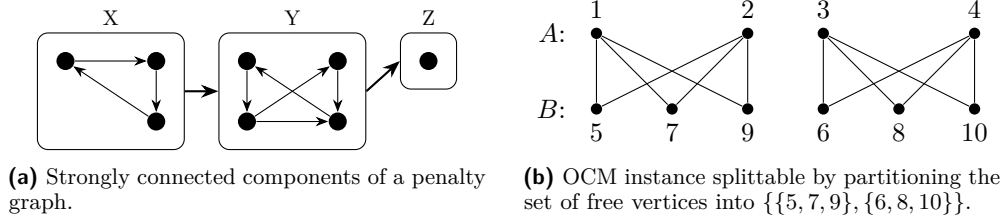
19th International Symposium on Parameterized and Exact Computation (IPEC 2024).

Editors: Édouard Bonnet and Paweł Rzażewski; Article No. 33; pp. 33:1–33:4

Leibniz International Proceedings in Informatics



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



■ **Figure 1** Graph splitting approaches.

2 Data Reduction Rules

The following methods are employed in the heuristic, exact and parameterized track. Before applying data reduction rules we try to split the OCM instance into several smaller instances. We can solve each strongly connected component of the instance’s penalty graph individually and concatenate their solutions in the topological order of the penalty graph’s strongly connected components (visualized in Figure 1a).

If the graph is large, then the above approach is infeasible. Instead, we try to split the graph by partitioning the free vertices B into non-empty subsets $\mathcal{B} = \{B_1, B_2, \dots, B_k\}$ such that there are no vertices $u, v, w \in A$ with u before v before w in the fixed order of A and sets $B_i, B_j \in \mathcal{B}$ with $u, w \in N(B_i)$ and $v \in N(B_j)$. In other words the neighborhood intervals of the elements of \mathcal{B} do not overlap in the set of fixed vertices. We can split a graph into the induced subgraphs of each partition element and potentially further split these subgraphs with the aforementioned splitting approach. An example OCM instance that can be split via partitioning is Figure 1b.

Additionally, we apply data reductions proposed by Dujmovic et al. [4]. In particular their rules **RR1**, **RR2**, **RRL01** in unmodified form and a modified version of their **RRlarge** rule that accounts not only for an upper bound but also for the trivial lower bound described by Nagamochi [12].

3 Heuristic Track

We use different approaches to find a heuristic solution depending on the size of the graph. For small and medium-sized graphs, the repeated application of our methods leads to better results, but on large graphs, even a single application may stress the resource limit. The first step in our heuristic for small to medium graphs is to compute an initial order with the median heuristic which was introduced by Eades and Wormald [5].

Local Search

Sifting. To improve an order, we use *sifting*, which was first introduced by Rudell [14]. One vertex is taken at a time and placed at the position in the order which minimizes the total number of crossings. We do this exhaustively for all free vertices of the graph, but choose the sequence in which we sift the vertices randomly. After applying sifting on the order computed by the median heuristic, we apply it on random orders and store the order with the fewest crossings.

Swapping. To escape local optima, we use *force swapping* if no better solution was found for 592 iterations¹ of using a random order with sifting. Force swapping is done by choosing two vertices and swapping their positions in the order. Then, we use sifting to improve the order while requiring that the relative position of the two vertices remains unchanged. We iterate through all vertices in a random order, swapping each vertex with its immediate right neighbor within the current best ordering. Subsequently, we increment the distance from the swapped vertices in each iteration by 9 until the distance is greater than 90.

Large Graphs

For large graphs, we first compute an order with the median heuristic and try to improve it by swapping neighboring vertices in the order until 10% of the available time is used. We repeat the same with the barycenter heuristic [15]. Then, we use a variation of sifting to improve the best order found so far. Instead of checking all possible positions for each vertex in the order, we skip the vertex if the total number of crossings increases by 20,000 or more. We sift the vertices until the time limit is reached.

4 Exact Track & Parameterized Track

For the exact solver we mainly use the idea described in the introduction to translate our OCM to an FAS instance. Our approach for solving the FAS instance is based on an algorithm by Grötschel et al. [1]. The idea is to solve the FAS problem iteratively, by only looking at a subset \mathcal{C} of all cycles from the input graph G and add cycles until one can guarantee that the solution, when only looking at the cycles \mathcal{C} , is also optimal for the initial graph.

ILP. To now solve a partial instance of feedback arc set with the cycles \mathcal{C} , we use the following ILP formulation, which was widely used to solve FAS [1, 8, 13]:

$$\begin{aligned} \min \quad & \sum_{e \in E} w(e) \cdot y_e \\ \text{s.t.} \quad & y_e \in \{0, 1\} && \text{for all } e \in E \\ & \sum_{e \in C} y_e \geq 1 && \text{for all } C \in \mathcal{C} \end{aligned}$$

Where we have a variable y_e for each $e \in E$, which is 1 if and only if the edge e is part of an FAS. As a solver we use SCIP v9.0 [3] and together with row generation and a warm startup so it can use results from the previous iteration, when solving the next iteration with more cycles.

Branching. For graphs where the upper bound is smaller than 10, we use a branch & bound algorithm to solve FAS. The algorithm branches on the edges of a cycle, uses a meta-heuristic by Lan et al. [9] as upper bound and a packing lower bound of the cycles.


The main reason for the use of the branching comes into play for the parameterized track. Here, our graphs decompose into a lot of small components using the splitting described in Section 2. When we then used SCIP to solve FAS, there was a large overhead in calling the ILP compared to the time the ILP needed to solve the instance.

¹ The exact values were found with SMAC3 using a subset of the public instances for training [10]

References

- 1 Ali Baharev, Hermann Schichl, Arnold Neumaier, and Tobias Achterberg. An exact method for the minimum feedback arc set problem. *ACM J. Exp. Algorithmics*, 26:1.4:1–1.4:28, 2021. doi:10.1145/3446429.
- 2 Kimon Boehmer, Lukas Lee George, Fanny Hause, and Jesse Palarus. Arcee, June 2024. Software, swbId: swb:1.dir:4a1810b816bec954e0780d591b0c79276ac8f285 (visited on 2024-11-27). doi:10.5281/zenodo.12171404.
- 3 Suresh Bolusani, Mathieu Besançon, Ksenia Bestuzheva, Antonia Chmiela, João Dionísio, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Mohammed Ghannam, Ambros Gleixner, Christoph Graczyk, Katrin Halbig, Ivo Hedtke, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Dominik Kamp, Thorsten Koch, Kevin Kofler, Jurgen Lentz, Julian Manns, Gioni Mexi, Erik Mühmer, Marc E. Pfetsch, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Mark Turner, Stefan Vigerske, Dieter Weninger, and Lixing Xu. The SCIP Optimization Suite 9.0. Technical report, Optimization Online, February 2024. URL: <https://optimization-online.org/2024/02/the-scip-optimization-suite-9-0/>.
- 4 Vida Dujmovic, Henning Fernau, and Michael Kaufmann. Fixed parameter algorithms for one-sided crossing minimization revisited. *J. Discrete Algorithms*, 6(2):313–323, 2008. doi:10.1016/J.JDA.2006.12.008.
- 5 Peter Eades and Nicholas C Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11:379–403, 1994. doi:10.1007/BF01187020.
- 6 Michael Jünger and Petra Mutzel. 2-layer straightline crossing minimization: Performance of exact and heuristic algorithms. In *Graph algorithms and applications i*, pages 3–27. World Scientific, 2002.
- 7 Yasuaki Kobayashi and Hisao Tamaki. A fast and simple subexponential fixed parameter algorithm for one-sided crossing minimization. *Algorithmica*, 72(3):778–790, 2015. doi:10.1007/S00453-014-9872-X.
- 8 Pichayapan Kongpanna, Deenesh K. Babi, Varong Pavarajarn, Suttichai Assabumrungrat, and Rafiqul Gani. Systematic methods and tools for design of sustainable chemical processes for co₂ utilization. *Comput. Chem. Eng.*, 87:125–144, 2016. doi:10.1016/J.COMPCHEMENG.2016.01.006.
- 9 Guanghui Lan, Gail W. DePuy, and Gary E. Whitehouse. An effective and simple heuristic for the set covering problem. *Eur. J. Oper. Res.*, 176(3):1387–1403, 2007. doi:10.1016/J.EJOR.2005.09.028.
- 10 Marius Lindauer, Katharina Eggenberger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. Smac3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022. URL: <http://jmlr.org/papers/v23/21-0888.html>.
- 11 Christian Matuszewski, Robby Schönfeld, and Paul Molitor. Using sifting for k-layer straightline crossing minimization. In Jan Kratochvíl, editor, *Graph Drawing, 7th International Symposium, GD'99, Střirín Castle, Czech Republic, September 1999, Proceedings*, volume 1731 of *Lecture Notes in Computer Science*, pages 217–224. Springer, 1999. doi:10.1007/3-540-46648-7_22.
- 12 Hiroshi Nagamochi. On the one-sided crossing minimization in a bipartite graph with large degrees. *Theor. Comput. Sci.*, 332(1-3):417–446, 2005. doi:10.1016/J.TCS.2004.10.042.
- 13 TK Pho and L Lapidus. Topics in computer-aided design: Part i. an optimum tearing algorithm for recycle systems. *AIChE Journal*, 19(6):1170–1181, 1973.
- 14 R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proceedings of 1993 International Conference on Computer Aided Design (ICCAD)*, pages 42–47, 1993. doi:10.1109/ICCAD.1993.580029.
- 15 Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiro Toda. Methods for visual understanding of hierarchical system structures. *IEEE Trans. Syst. Man Cybern.*, 11(2):109–125, 1981. doi:10.1109/TSMC.1981.4308636.

PACE Solver Description: LUNCH – Linear Uncrossing Heuristics

Kenneth Langedal¹ ✉ 

University of Bergen, Norway

Matthias Bentert ✉

University of Bergen, Norway

Thorgal Blanco ✉ 

University of Bergen, Norway

Pål Grønås Drange ✉ 

University of Bergen, Norway

Abstract

The 2024 PACE challenge is on ONE-SIDED CROSSING MINIMIZATION: Given a bipartite graph with one fixed and one free layer, compute an ordering of the vertices in the free layer that minimizes the number of edge crossings in a straight-line drawing of the graph. Here, we briefly describe our exact, parameterized, and heuristic submissions. The main contribution is an efficient reduction to a weighted version of DIRECTED FEEDBACK ARC SET, allowing us to detect subproblems that can be solved independently.

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

Keywords and phrases graph drawing, feedback arc set, algorithm engineering

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.34

Supplementary Material

Software (Source Code): <https://github.com/KennethLangedal/PACE2024-UiB> [1]

archived at `swh:1:dir:1acb0d8737c376662f00d81d2b831f20ccca0d21`

1 Introduction

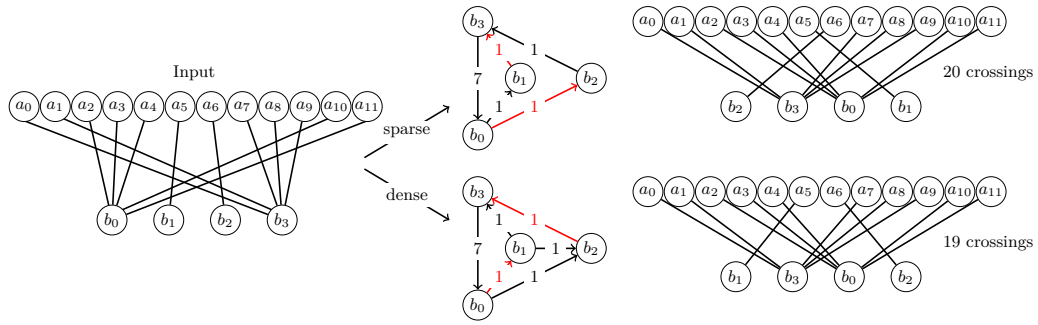
Let $G = ((A \uplus B), E)$ be an undirected bipartite graph with vertex partition (A, B) . In ONE-SIDED CROSSING MINIMIZATION (OCM), an ordering τ of A is given and the task is to compute an ordering π for B that minimizes the edge crossings in a straight-line drawing of G . The number of edge crossings for a linear ordering of B can be computed by comparing pairs of vertices in B separately. Let $c_{u,v}$ denote the number of edge crossings between u and v when u is placed before v . The cost of an ordering π of B is $\sum_{u,v \in B | \pi(u) < \pi(v)} c_{u,v}$. We will denote by ℓ_u and r_u the leftmost and rightmost neighbors of u with respect to τ in A .

2 Preprocessing

The first step of both our exact and heuristic solvers is to reduce OCM to the weighted version of DIRECTED FEEDBACK ARC SET (DFAS). The instance for the latter problem will contain a vertex v' for each vertex $v \in B$. For each pair u, v of vertices in B , compute $c_{u,v}$ and $c_{v,u}$. Add a directed arc (u', v') of weight $c_{v,u} - c_{u,v}$ if $c_{u,v} \leq c_{v,u}$. Otherwise, add the arc (v', u') with weight $c_{u,v} - c_{v,u}$. Assume without loss of generality that $c_{u,v} < c_{v,u}$. Then, if we place u' before v' (which corresponds to placing u before v), then we pay $c_{u,v}$, and if we

¹ Corresponding author





■ **Figure 1** Example showing how an optimal solution to a sparse component can be suboptimal. The two DFAS instances in the middle have optimal solutions shown by red edges. When lifting the solution back to the OCM problem, the sparse component places b_1 and b_2 in the wrong order.

place u' after v' , then $c_{v,u} = c_{u,v} + (c_{v,u} - c_{u,v})$. Since we are only interested in minimizing the number of crossings and the difference between these two options is preserved, we get an equivalent instance of DIRECTED FEEDBACK ARC SET.

One very effective reduction in the constructed DFAS instance is that edges between strongly connected components can be deleted since these edges are not part of any cycles. Furthermore, strongly connected components can be solved independently. We initially construct a sparse instance to speed up the search for strongly connected components. Two types of edges will be ignored at this stage: (1) edges (u', v') where $c_{u,v} = 0$ and (2) edges (u', v') where $c_{u,v} = c_{v,u}$. The construction of this sparse instance works as follows:

```

1:  $\pi \leftarrow \{\ell_{u_0} \leq \ell_{u_1} \dots \leq \ell_{u_{|B|-1}}\}$ 
2: for  $i = 0, \dots, |B| - 1$  do
3:   for  $j = i + 1, \dots, |B| - 1$  do
4:      $u \leftarrow \pi[i], v \leftarrow \pi[j]$ 
5:     if  $r_u \leq \ell_v$  then
6:       break ▷ all remaining pairs have  $c_{u,v} \neq 0$  and  $c_{v,u} \neq 0$ 
7:     if  $c_{uv} < c_{vu}$  then
8:       Add edge  $(u', v')$  with weight  $c_{vu} - c_{uv}$ 
9:     if  $c_{uv} > c_{vu}$  then
10:      Add edge  $(v', u')$  with weight  $c_{uv} - c_{vu}$ 
    
```

The break statement on lines 5 and 6 (highlighted in red) is what makes this procedure worthwhile. When we first encounter a pair of vertices that have $c_{u,v} = 0$, we know that every remaining u, v pair in the inner most loop must also have $c_{u,v} = 0$, since ℓ_v is increasing. This improvement had large effects on many of the test instances. We will show next that looking for strongly connected components in this graph is safe. However, such edges within a strongly connected component cannot be ignored as the example in Figure 1 shows.

We proceed by showing that ignoring edges between different strongly connected components in the described graph G' is safe. To this end, we consider three types of arcs (represented by colors). We color an arc (u', v') as follows. If $c_{u,v} = 0$, then we color the arc red. If $c_{u,v} = c_{v,u}$, then we color the arc green. All other arcs (those that we do not ignore) are blue. We will repeatedly make use of the following lemma.

▶ **Lemma 1.** *If (u', v') is a red arc and (v', w') is a blue arc, then (w', u') cannot be a green or blue arc. The same holds if (u', v') is blue and (v', w') is red.*

Proof. We will only show the statement for (u', v') being red as the other case is symmetric. Assume towards a contradiction that (w', u') is a green or blue edge. Note that (u', v') being red implies that $\ell_v > r_u$. Let $L, R \subseteq A$ be the set of neighbors of w to the left/right of r_u , that is, $a \in A$ belongs to L if $\tau(a) < \tau(r_u)$ and a belongs to R if $\tau(a) > \tau(r_u)$. Note that $c_{w,u} \geq |N(u)| \cdot |R|$. Since (w', u') is green or blue, it holds that $c_{u,w} \geq c_{w,u} \geq |N(u)| \cdot |R|$. This implies that $|L| \geq |R|$. However, since $\ell_v > r_u$, this implies also that $c_{v,w} \geq |L| \cdot |N(v)| \geq |R| \cdot |N(v)| c_{w,v} \geq c_{w,v}$ which contradicts the fact that (v', w') is a blue arc. \blacktriangleleft

We need to show that there is no directed cycle consisting of arcs with positive weights that contains a red or green arc (u', v') where u' and v' belong to different strongly connected components in the graph induced by all blue arcs. Note that since all green arcs have weight 0, they can never be part of such a cycle. So assume towards a contradiction that there exists a directed cycle C that contains a red arc (u', v') where u' and v' belong to different strongly connected components in the graph induced by all blue arcs and all arcs in C are red or blue. We assume without loss of generality that C is the shortest (in terms of number of vertices) such cycle. Let $C = (v' = w'_0, w'_1, \dots, w'_c = u')$. Note that by definition (w'_i, w'_{i+1}) exists and is either a blue or a red arc. Note that if any red or blue arc (w'_i, w'_j) with $i < j - 1$ exists, then this is a shortcut and contradicts the fact that C is a shortest cycle. Moreover, any red arc (w'_i, w'_j) with $i > j$ (other than $i = c$ and $j = 0$) would also imply a shorter cycle than C .

Next, assume that some arc (w'_i, w'_{i+1}) is red. Now any blue arc (w'_j, w'_k) with $j > i$ and $k \leq i$ would contradict the fact that C is a shortest cycle. Hence, all such arcs are green. Now consider the arc (w'_{i-1}, w'_i) (where $w'_{i-1} = u'$ if $i = 0$). If this arc is red, then $r_{w_{i-1}} \leq \ell_{w_i} \leq r_{w_i} \leq \ell_{w_{i+1}}$, showing that (w'_{i-1}, w'_{i+1}) is a red arc, a contradiction. Hence, the arc is blue (and $w'_{i-1} \neq u'$). However, now (w'_{i-1}, w'_i) is blue, (w'_i, w'_{i+1}) is blue, and (w'_{i+1}, w'_{i-1}) is green as $i + 1 > i$ and $i - 1 \leq i$ shows that the arc cannot be blue and it can also not be red as shown above. This contradicts Lemma 1 and shows that all arcs (w'_i, w'_{i+1}) are blue.

To conclude the argument that C cannot exist, consider the pair $\{u', w'_1\}$. By Lemma 1, there cannot be a green arc between the two. We now consider two cases, $c = 2$ (that is, C consist of u', v' , and w'_1) or $c > 2$. If $c = 2$, then (w'_1, u') is a blue arc and this contradicts Lemma 1. If $c > 2$, then there cannot be an arc (w'_1, u') as this would contradict the fact that C is a shortest cycle. Hence, in this case (u', w') is a blue arc (it cannot be red arc as this would mean that we could exclude v' from C to get a shorter cycle through a red arc). Hence, w'_1 and u' (and in fact all w'_i other than v') belong to the same strongly connected component in G' . Now consider any vertex w'_i with $i \notin \{0, 1, c\}$. As shown above, the arc (v', w'_i) is not red or blue. Hence, the arc (w'_i, v') is red, blue, or green. It cannot be red as this would result in a shorter cycle than C through this arc. It can also not be blue as this would mean that v' is in the same strongly connected component in G' as w'_i (and therefore as u'). Thus, all such edges are green but this means that the edge (v', w'_{c-1}) is green, a final contradiction to Lemma 1.

3 Tiny components

We use a dynamic-programming algorithm to solve tiny components quickly when the number of vertices is at most twenty. It could also be used to solve larger components, but our dedicated exact solver will solve larger instances faster. The algorithm is based on the dynamic-programming algorithm for DFAS by Lawler [2] and can simply be described by the recurrence

$$\text{dp}(\emptyset) = 0 \quad | \quad \text{dp}(S) = \min_{u \in S} \text{dp}(S \setminus \{u\}) + \text{deg}(u, S)$$

where $\text{deg}(u, S)$ is the number of edges going from u to S .

4 Heuristic

Our heuristic mainly relies on the cutting technique introduced by Park and Akers [3]. Instead of explicitly breaking cycles, consider the DFAS problem as finding an ordering of the vertices that minimizes the number of edges going from right to left. Now, the cutting technique used by Park and Akers searches every continuous subgraph in the current ordering, and any cut within each subgraph. If, at any time, the number of edges going backward across the cut is larger than the number of edges going forward across the cut, we swap the vertices before and after the cut. While this procedure seems like it would take $O(n^4)$ time, it can be done in $O(n^3)$ time using a cut matrix [3]. To speed up the computation further, we also limit the distance the cut can be from any side of the subgraph.

In several instances, the cutting idea is still too slow, so we only use it after cheaper greedy improvements fail to make progress. We randomize the current solution to escape local minimums by swapping 1–3 random pairs of vertices while always returning to the best solution if the next local minimum was worse. In very few cases, the graphs are so large that reducing to DFAS is impossible without exceeding the memory or time limits. In these cases, we only use greedy improvements while repeatedly computing $c_{u,v}$ when needed.

5 Exact

We first run our heuristic for each large component to get an upper bound on the DFAS solution. Then, our exact method starts by enumerating all short cycles in the graph (cycles with at most four vertices). Then, create a MAXSAT instance where each cycle is a hard constraint requiring at least one of the edges in the cycle to be picked. Every edge also has its own soft constraint with the weight of the edge. We then solve this MAXSAT instance using the solver UWrMaxSat [4]. There are now two termination conditions: (1) after removing the edges from the MAXSAT solution, the resulting graph is acyclic, and (2) the cost of the MAXSAT solution is equal to our upper bound. In both cases, we have an optimal solution. In the first, the edges removed in our MAXSAT instance also make an optimal solution to the DFAS problem. In the second case, we know our upper-bound solution was optimal. Otherwise, the solver proceeds by temporarily removing the edges in the latest solution from the MAXSAT instance. Since this graph is not acyclic, we can find new cycles using a depth first search. We add these cycles to our MAXSAT instance and repeat until we hit one of the two termination conditions mentioned above.

References

- 1 Kenneth Langedal. KennethLangedal/PACE2024-UiB: pace-2024, June 2024. doi:10.5281/zenodo.11540761.
- 2 Eugene L. Lawler. A comment on minimum feedback arc sets. *IEEE Transactions on Circuit Theory*, 11(2):296–297, 1964.
- 3 Sungju Park and Sheldon B. Akers. An efficient method for finding a minimal feedback arc set in directed graphs. In *1992 IEEE International Symposium on Circuits and Systems*, volume 4, pages 1863–1866. IEEE, 1992.
- 4 Marek Piotrów. UWrMaxSat: Efficient solver for MaxSAT and pseudo-boolean problems. In *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 132–136. IEEE, 2020. doi:10.1109/ICTAI50040.2020.00031.

PACE Solver Description: OCMu64, a Solver for One-Sided Crossing Minimization

Ragnar Groot Koerkamp   

ETH Zurich, Switzerland

Mees de Vries  

Unaffiliated, The Netherlands

Abstract

Given a bipartite graph (A, B) , the *one-sided crossing minimization* (OCM) problem is to find an ordering of the vertices of B that minimizes the number of edge crossings when drawn in the plane.

We introduce the novel *strongly fixed*, *practically fixed*, and *practically glued* reductions that maximally generalize some existing reductions. We apply these in our exact solver **OCMu64**, that directly uses branch-and-bound on the ordering of the vertices of B and does not depend on ILP or SAT solvers.

2012 ACM Subject Classification Theory of computation \rightarrow Mathematical optimization; Theory of computation \rightarrow Computational geometry

Keywords and phrases Graph drawing, crossing number, branch and bound

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.35

Supplementary Material *Software (Source Code)*: <https://github.com/mjdv/ocmu64> [6]

Software (Source Code): <https://doi.org/10.5281/zenodo.11671980> [7]

1 Introduction

The 2024 edition of PACE, an annual optimization challenge, considers the *one-sided crossing minimization* problem, defined as follows. Given is a bipartite graph (A, B) that is drawn in the plane at points $(i, 0)$ and $(j, 1)$ for components A and B respectively. The ordering of A is fixed, and the goal is to find an ordering of B that minimizes the number of crossings when edges are drawn as straight lines. We introduce some new reductions and give an overview of our algorithm. Proofs are brief or omitted due to lack of space.

2 Definitions

We use $<$ to compare vertices in A in their fixed ordering. We generalize to weighted graphs for the proof of Lemma 3: a node $u \in B$ is taken to be a function $u : A \rightarrow \mathbb{R}^{\geq 0}$, where weight 0 represents an absent edge. Write $N(u) = \text{supp}(u) \subseteq A$ for the set of neighbors of u . We write $W_u = \sum_{a \in A} u(a)$ for the total weight of u , and set $\bar{u} = u/W_u$. We think of \bar{u} as a probability distribution, and also consider the cumulative distribution function $F_{\bar{u}}(b) = \sum_{a \leq b} \bar{u}(a)$. We write $c(u, v) = \sum_{(a,b) \in A^2} u(a)v(b)[a > b]$; in the unweighted case, this is the *crossing number*, the number of crossings between edges incident to u and v when u is drawn before v . For $X, Y \subseteq B$ we set $c(X, Y) = \sum_{x \in X} \sum_{y \in Y} c(x, y)$ for the cost of ordering all vertices of X before all vertices of Y . More generally, $c(X, Y, Z) = c(X, Y) + c(X, Z) + c(Y, Z)$. We also consider the *reduced cost* $r(X, Y) = c(X, Y) - c(Y, X)$, which is negative when X is better before Y and positive when X is better after Y .

We write $u \prec v$ when u must come before v in all minimal solutions, and say that (u, v) is a *fixed* pair.



© Ragnar Groot Koerkamp and Mees de Vries;
licensed under Creative Commons License CC-BY 4.0

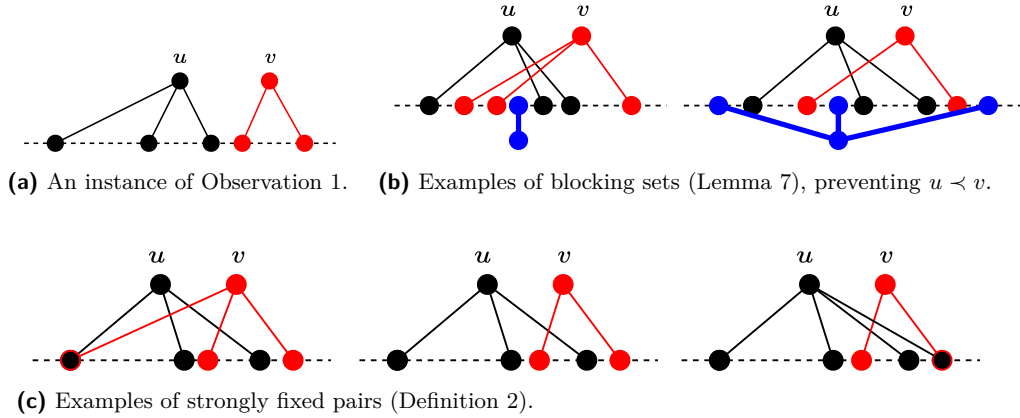
19th International Symposium on Parameterized and Exact Computation (IPEC 2024).

Editors: Édouard Bonnet and Paweł Rzażewski; Article No. 35; pp. 35:1–35:5

Leibniz International Proceedings in Informatics



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



■ **Figure 1** Various examples of $u, v \in B$, whose neighbors neighbours in A (on the dashed line) have fixed positions.

3 Methods

3.1 Reductions

Fixed pairs. To reduce the search space of all possible orderings of B , it is crucial to automatically find as many fixed pairs in B as possible. Ideally, one would be able to determine whether $u < v$ by inspecting only u, v . For example, the following result, shown in Figure 1a is well known [2, Lemma 1], [1, RR1].

► **Observation 1.** *Let $u, v \in B$. When $\max N(u) \leq \min N(v)$ and $\bar{u} \neq \bar{v}$, then $u < v$.*

We give a much stronger version of Observation 1, with some examples in Figure 1c. In fact, Lemma 3 is the strongest generalization possible when only considering u and v themselves, without further inspection of A or the other elements of B (see Remark 6). This also generalizes RR3 of [1].

► **Definition 2 (Strongly fixed pair).** *We call $u, v \in B$ a strongly fixed pair if for every $b \in A$ we have $F_{\bar{u}}(b) \geq F_{\bar{v}}(b)$, and at least one of these inequalities is strict. Note that this implies $r(u, v) < 0$.*

► **Lemma 3.** *When u, v is a strongly fixed pair, then for any $w : A \rightarrow \mathbb{R}^{\geq 0}$ we have*

$$r(w, \bar{v}) \leq r(w, \bar{u}).$$

Proof. Consider the least element $a_0 \in A$ such that $\bar{u}(a_0) \neq \bar{v}(a_0)$. We must have $\bar{u}(a_0) > \bar{v}(a_0)$. Now consider the transformation of \bar{u} which “moves” $\delta := \bar{u}(a_0) - \bar{v}(a_0)$ of weight from a_0 to its successor $a_1 \in A$, and call this transformed function \bar{u}' . Then

$$r(w, \bar{u}') = r(w, \bar{u}) - \delta w(a_0) - \delta w(a_1) \leq r(w, \bar{u}).$$

Since \bar{v} is obtained from \bar{u} by a sequence of such transformations, the inequality follows. ◀

► **Lemma 4.** *If (u, v) is strongly fixed, then $u < v$.*

Proof. Suppose towards a contradiction that $v < x_0 < \dots < x_k < u$ is part of an optimal solution. Write $X = \sum_i x_i$ for the combined function. Then by assumption $r(X, u) \leq 0$, and therefore $r(X, v) = W_v r(X, \bar{v}) \leq W_v r(X, \bar{u}) = W_v/W_u \cdot r(X, u) \leq 0$. But then $c(X, u, v) < c(X, v, u) \leq c(v, X, u)$, which contradicts (v, X, u) being optimal. ◀

► **Remark 5.** Consider $u \neq v$ from the original, unweighted problem, taking only values 0, 1. Let $n = |N(u)|$, $m = |N(v)|$, and consider both as ordered lists. Then u, v are strongly fixed if and only if for all $0 \leq i < n$, $N(u)_i \leq N(v)_{\lfloor i \cdot m/n \rfloor}$ and at least one of the inequalities is strict, which is how we check in practice whether u, v is strongly fixed.

► **Remark 6.** Suppose that u, v with $\bar{u} \neq \bar{v}$ are not strongly fixed, and let $b \in A$ be such that $F_{\bar{u}}(b) < F_{\bar{v}}(b)$. We would like to construct a node $X \in B$ such that $c(v, X, u) < \min(c(X, u, v), c(u, v, X))$, so that $v \prec u$. Suppose that there are nodes $l, m, r \in A$, with m sitting between b and its successor in A , and $l < (\text{supp}(u) \cup \text{supp}(v)) < r$ (see Figure 1b). Then let X be connected to l, m, r . By choosing the weights of X appropriately, we can let both $r(X, v) = W_v r(X, \bar{v})$ and $r(u, X) = W_u r(\bar{u}, X)$ grow arbitrarily large. Since we can scale up the weights of X , it suffices to make $r(X, \bar{v})$ and $r(\bar{u}, X)$ simultaneously positive.

Now $r(X, \bar{v}) = (2F_{\bar{v}}(b) - 1)X(m) + (X(r) - X(l))$, and $r(\bar{u}, X) = (1 - 2F_{\bar{u}}(b))X(m) - (X(r) - X(l))$. By choosing the weights $X(r), X(l)$ appropriately, we can make these two values equal, so it suffices if their *average* is positive, and indeed

$$(r(X, \bar{v}) + r(\bar{u}, X))/2 = X(m)(F_{\bar{v}}(b) - F_{\bar{u}}(b)) > 0.$$

This implies $c(v, X, u) < \min(c(X, u, v), c(u, v, X))$, and thus demonstrates that if one wants to show that $u \prec v$, when u, v are not strongly fixed, one must consider features of the graph other than u, v and their neighbours. In other words, Lemma 3 is optimal. Note that this remark also applies in the unweighted case, by taking l, m, r to be sets of nodes rather than single nodes with weighted edges.

Although such a node X may exist in theory, it does not have to exist in the actual set B , motivating the following definition that generalizes RRLO2 of [1].

► **Lemma 7.** *Suppose $r(u, v) < 0$. A blocking set $X \subseteq B - \{u, v\}$ is a set such that $c(v, X, u) \leq \min(c(v, u, X), (X, v, u))$. If there is no blocking set for (u, v) , we call it a practically fixed pair, and $u \prec v$.*

In practice, such a blocking set X can be found, if one exists, using a knapsack-like algorithm: for each $x \in B - \{u, v\}$, add a point $P_x = (r(v, x), r(x, u))$, and search for a subset summing to $\leq (r(u, v), r(u, v))$. Figure 1b shows some examples.

Note that we do not require (v, X, u) to be a true local minimum, since we do not consider interactions between vertices in X , as that would make ruling out the existence of such sets much harder.

► **Remark 8 (Weak variants).** It is also possible to consider *weak* variants of the above lemmas that only imply that $u < v$ in *some* optimal solution. This requires careful handling of cycles like $u \preceq v \preceq w \preceq u$.

Gluing. We now turn our attention to *gluing*, i.e., proving that two vertices u and v always go right next to each other, and we can treat them as a single vertex. First, let us see that we cannot get a “strong gluing” result analogous to Lemma 4.

► **Remark 9.** When $N(u) = N(v)$ in the unweighted case, or more generally $\bar{u} = \bar{v}$, we can glue u and v . Otherwise when $r(u, v) \leq 0$, there is an $X : A \rightarrow \mathbb{R}^{\geq 0}$ such that (u, X, v) is strictly better than (u, v, X) and (X, u, v) .

► **Lemma 10 (Practical gluing).** *Let u and v satisfy $r(u, v) \leq 0$. A non-empty subset $X \subseteq B - \{u, v\}$ is blocking when $c(u, X, v) \leq \min(c(u, v, X), c(X, u, v))$. If there is no blocking set, then we can glue u, v .*

Again such sets X can be found or proven to not exist using a knapsack algorithm: add points $P_x = (r(u, x), r(x, v))$ and search for a non-empty set summing to $\leq (0, 0)$.

Let us also mention this gluing-like reduction: *gluing to the front*, implied by [1, RRL01].

► **Lemma 11** (Greedy). *When $r(u, x) \leq 0$ for all $x \in B$, there is a solution that starts with u .*

► **Remark 12** (Tail variants). Our branch-and-bound method fixes vertices of the solution from left to right. Thus, at each step Lemmas 7 and 10 can be applied to just the *tail*.

3.2 Branch-and-bound

Our solver OCMu64 is based on a standard branch-and-bound on the ordering of the solution. We start with fixed prefix $P = ()$ and tail $T = B$, and in each step we try (a subset of) all vertices in T as the next vertex appended to P . In a preprocessing step we compute the trivial lower bound $S_0 = \sum_{u,v} \min(c(u, v), c(v, u))$ [5, Lemma 4][2, Fact 3] on the score. We keep track of the score S_P of the prefix and $S_{PT} = c(P, T)$ of prefix-tail intersections, and abort when this score goes above the best solution found so far. The *excess* of a tail is its optimal score minus the trivial lower bound. We do a number of optimizations.

Graph simplification We drop degree-0 vertices, merge identical vertices, and split the graph into *independent* components [2, Corollary 2] when possible. We find an initial solution using the median heuristic [3, 5] and a local search that tries to move slices and optimally insert them [8, 4], and re-label all nodes accordingly to make memory accesses more efficient.

Fixed pairs We find all strongly fixed pairs and store them. For the exact track we also find practically fixed pairs. Instances for the parameterized track are simple enough that the overhead was not worth it. Also for each tail we search for new “tail-local” practically fixed pairs. In each state, we only try vertices $u \in T$ not fixed by another $v \in T$.

Gluing We use the greedy strategy of Lemma 11. Our implementation of Lemma 10 contained a bug, so we did not use this. (Also benefits seemed limited.)

Tail cache In each step, we search for the longest suffix of T that was seen before, and reuse (the lower bound on) its excess. We also cache the tail-local practically fixed pairs.

Optimal insert Instead of simply appending u to P , we insert it in the optimal position. Note that the implementation is tricky because it interacts in complicated ways with the caching of results for each tail.

References

- 1 Vida Dujmovic, Henning Fernau, and Michael Kaufmann. Fixed parameter algorithms for one-sided crossing minimization revisited. *Journal of Discrete Algorithms*, 6(2):313–323, June 2008. doi:10.1016/j.jda.2006.12.008.
- 2 Vida Dujmovic and Sue Whitesides. An efficient fixed parameter tractable algorithm for 1-sided crossing minimization. *Algorithmica*, 40(1):15–31, April 2004. doi:10.1007/s00453-004-1093-2.
- 3 P. Eades and N.C. Wormald. *The Median Heuristic for Drawing 2-layered Networks*. Technical report. University of Queensland, Department of Computer Science, 1986.
- 4 Peter Eades and David Kelly. Heuristics for reducing crossings in 2-layered networks. *Ars Combinatoria*, 21(A):89–98, 1986.
- 5 Peter Eades and Nicholas C. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11(4):379–403, April 1994. doi:10.1007/bf01187020.
- 6 Ragnar Groot Koerkamp and Mees de Vries. OCMu64. Software (visited on 2024-11-28). URL: <https://github.com/mjdv/ocmu64>, doi:10.4230/artifacts.22525.

- 7 Ragnar Groot Koerkamp and Mees de Vries. OCMu64. Software (visited on 2024-11-28). doi:10.5281/zenodo.11671980.
- 8 Erkki Mäkinen. Experiments on drawing 2-level hierarchical graphs. *International Journal of Computer Mathematics*, 36(3-4):175–181, January 1990. doi:10.1080/00207169008803921.

