16th International Symposium on Parameterized and Exact Computation

IPEC 2021, September 8–10, 2021, Lisbon, Portugal

Edited by

Petr A. Golovach

Meirav Zehavi

Editors

Petr A. Golovach
University of Bergen, Norway
petr.golovach@ii.uib.no

Meirav Zehavi
Ben-Gurion University of the Negev, Israel
meiravze@bgu.ac.il

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

ISBN 978-3-95977-216-7

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/978-3-95977-216-7.

Publication date
November, 2021

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

Digital Object Identifier: 10.4230/LIPIcs.IPEC.2021.0

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.

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Regular Papers

16th International Symposium on Parameterized and Exact Computation (IPEC 2021).
Editors: Petr A. Golovach and Meirav Zehavi
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

PACE Solver Descriptions

Preface

This volume contains the papers presented at IPEC 2021: the 16th International Symposium on Parameterized and Exact Computation. IPEC 2021 was held on September 8–10. It was a part of the ALGO 2021 congress, and took place virtually due to the Covid 19 pandemic.

The International Symposium on Parameterized and Exact Computation (IPEC, formerly IWPEC) is a series of international symposia covering research in all aspects of parameterized and exact algorithms and complexity. Started in 2004 as a biennial workshop, it became an annual event in 2009.

Previous iterations of I(W)PEC	
2004	Bergen, Norway
2006	Zürich, Switzerland
2008	Victoria, Canada
2009	Copenhagen, Denmark
2010	Chennai, India
2011	Saarbrücken, Germany
2012	Lubljana, Slovenia
2013	Sophia Antipolis, France
2014	Wrocław, Poland
2015	Patras, Greece
2016	Aarhus, Denmark
2017	Vienna, Austria
2018	Helsinki, Finland
2019	Münich, Germany
2020	Hong Kong, China

In response to the call for papers, 52 abstracts were submitted, which led to 48 papers considered by the program committee. Each considered submission received at least 3 reviews. The reviews came from the 18 members of the program committee and from 61 external reviewers, together contributing 150 reviews. The program committee held electronic meetings through the EasyChair platform. In the end, the program committee selected 25 of the submissions for presentation at the symposium and inclusion in these proceedings.

The Best Paper Award was given jointly to two co-winners:

- Guillaume Ducoffe, for his paper *Maximum Matching in Almost Linear Time on the Graphs of Bounded Clique-width*;
- Shaohua Li and Marcin Pilipczuk, for their paper *Hardness of Metric Dimension in Graphs of Constant Treewidth*.

The Best Student Paper Award was given to

- Vít Jelínek, Michal Opler, Jakub Pekárek, for their paper *Long Paths Make Pattern-counting Hard, and Deep Trees Make it Harder*.

IPEC 2021 hosted an award ceremony with a plenary talk for the 2021 EATCS-IPEC Nerode Prize for outstanding papers in the area of multivariate algorithmics. The Nerode prize committee consisted of Fedor V. Fomin, Anuj Dawar, and Virginia Vassilevska Williams. They awarded the prize to

- Cristian Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, Frank Stephan, for their paper *Deciding Parity Games in Quasi-polynomial Time* from STOC 2017.

Bakhadyr Khoussainov and Frank Stephan presented an invited talk.

IPEC 2021 also invited Édouard Bonnet to present a tutorial on twin-width of graphs. Finally, IPEC 2021 hosted the award ceremony of the sixth Parameterized Algorithms and Computational Experiments challenge, PACE. This yearly challenge was conceived in Fall 2015 to deepen the relationship between parameterized algorithms and practice. These proceedings contain a report by Leon Kellerhals, Tomohiro Koana, André Nichterlein and Philipp Zschoche on the 2021 PACE challenge and brief communications of the winners about their solvers.

16th International Symposium on Parameterized and Exact Computation (IPEC 2021).
Editors: Petr A. Golovach and Meirav Zehavi
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

We would like to thank the program committee, together with the external reviewers, for their commitment in the difficult paper selection process. We also thank all the authors who submitted their work for consideration. We are grateful to the local organizers of ALGO 2021 for their work on the local arrangements.

Petr A. Golovach and Meirav Zehavi
Bergen and Beer-Sheva, October 2021

Program Committees

IPEC 2021 Program Committee

Akanksha Agrawal
IIT Madras
India

Karl Bringmann
Saarland University
Germany

Jiehua Chen
Vienna University of Technology
Austria

Vincent Cohen-Addad
Google Zurich
Switzerland

Eduard Eiben
Royal Holloway, University of London
UK

Petr A. Golovach (co-chair)
University of Bergen
Norway

Karthik C. S.
New York University
USA

Martin Koutecký
Charles University
Czech Republic

William Lochet
University of Bergen
Norway

Jesper Nederlof
Utrecht University
Netherlands

Fahad Panolan
IIT Hyderabad
India

Ignasi Sau
LIRMM, Université de Montpellier, CNRS
France

Saket Saurabh
Indian Institute of Mathematical Sciences
India

Pascal Schweitzer
TU Kaiserslautern
Germany

Sebastian Siebertz
University of Bremen
Germany

Nimrod Talmon
Ben-Gurion University of the Negev
Israel

Dimitrios M. Thilikos
LIRMM, Université de Montpellier, CNRS
France

Meirav Zehavi (co-chair)
Ben-Gurion University of the Negev
Israel

PACE 2021 Program Committee

André Nichterlein (chair)
Technical University of Berlin
Germany

Leon Kellerhals
Technical University of Berlin
Germany

Tomohiro Koana
Technical University of Berlin
Germany

Philipp Zschoche
Technical University of Berlin
Germany

16th International Symposium on Parameterized and Exact Computation (IPEC 2021).
Editors: Petr A. Golovach and Meirav Zehavi
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

16th International Symposium on Parameterized and Exact Computation (IPEC 2021).
Editors: Petr A. Golovach and Meirav Zehavi
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

List of External Reviewers

Spyros Angelopoulos
N. R. Aravind
Emmanuel Arrighi
Valentin Bartier
Rémy Belmonte
Benjamin Bergougnoux
Hans L. Bodlaender
Édouard Bonnet
Nicolas Bousquet
Cornelius Brand
Robert Bredereck
Lijie Chen
Radu Curticapean
Argyrios Deligkas
Pål Grønås Drange
Henning Fernau
Nick Fischer
Fedor Fomin
Suprovat Ghoshal
Archontia Giannopoulou
Niels Grüttemeier
Sushmita Gupta
Shiri Heffetz
Tanmay Inamdar
Lars Jaffke
Pallavi Jain
Christos Kapoutsis
Eun Jung Kim
Dušan Knop
Euiwoong Lee
Erik Jan van Leeuwen
Bingkai Lin
Alexander Lindermayr
Raul Lopes
Diptapriyo Majumdar
Pasin Manurangsi
George Mertzios
Pranabendu Misra
Inbal Livni Navon
Karolina Okrasa
Mateus De Oliveira Oliveira
Miguel Romero Orth
Rutvik Page
Orr Paradise
Filip Pokrývka
Aditya Potukuchi
Nidhi Purohit
Venkatesh Raman
Malte Renken
Sanjukta Roy
Rudini Sampaio
Rik Sengupta
Roohani Sharma
Manuel Sorge
Krzysztof Sornat
Giannos Stamoulis
Sébastien Tavenas
Alexandre Vigny
Karol Węgrzycki
Sebastian Wiederrecht
Dimitris Zoros

16th International Symposium on Parameterized and Exact Computation (IPEC 2021).
Editors: Petr A. Golovach and Meirav Zehavi
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

List of Authors

Akanksha Agrawal (1, 2)
Indian Institute of Technology Madras, Chennai, India

Ravi Kiran Allumalla (2)
Indian Institute of Technology Madras, Chennai, India

Haozhe An (3)
University of Maryland, College Park, MD, USA

Aditya Anand (1)
Indian Institute of Technology Kharagpur, India

Júlio Araújo (4)
Departament of Mathematics, Federal University of Ceará, Fortaleza, Brazil

Vikraman Arvind (5)
The Institute of Mathematical Sciences (HBNI), Chennai, India

Jakub Balabán (6)
Masaryk University, Brno, Czech Republic

Max Bannach (7)
Universität zu Lübeck, Germany

Valentin Bartier (29, 33)
G-SCOP, Grenoble INP, Univ. Grenoble-Alpes, Grenoble, France

Gabriel Bathie (8, 29, 33)
École Normale Supérieure de Lyon, France

Thomas Bellitto (21)
Sorbonne Université, CNRS, LIP6 UMR 7606, Paris, France

Alexander Bille (28)
Computer Science, Philipps Universität Marburg, Germany

Thomas Bläsius (27, 31)
Karlsruhe Institute of Technology, Germany

Hans L. Bodlaender (9)
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Édouard Bonnet (10)
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Marin Bougeret (4)
LIRMM, Université de Montpellier, France

Nicolas Bousquet (8, 29, 33)
LIRIS, CNRS, Université Claude Bernard Lyon 1, Université de Lyon, France

Dominik Brandenstein (28)
Computer Science, Philipps Universität Marburg, Germany

Laurent Bulteau (11)
LIGM, CNRS, Univ Gustave Eiffel, F77454 Marne-la-vallée, France

Elisabet Burjons (12)
Department of Computer Science, RWTH Aachen, Germany

Victor Campos (4)
Departament of Computer Science, Federal University of Ceará, Fortaleza, Brazil

Yixin Cao (13)
Department of Computing, Hong Kong Polytechnic University, Hong Kong, China

Oscar Defrain (21)
Aix-Marseille Université, CNRS, LIS UMR 7020, Marseille, France

Varun Teja Dhanekula (2)
Indian Institute of Technology Madras, Chennai, India

Jona Dirks (30)
University of Bremen, Germany

Huib Donkers (14)
Eindhoven University of Technology, The Netherlands

Guillaume Ducoffe (15, 16)
National Institute for Research and Development in Informatics, Bucharest, Romania; University of Bucharest, Romania

Maël Dumas (17)
Univ. Orléans, INSA Centre Val de Loire, LIFO EA 4022, F-45067 Orléans, France

Andreas Emil Feldmann (18)
Department of Applied Mathematics, Charles University, Prague, Czech Republic

Philipp Fischbeck (27, 31)
Hasso Plattner Institute, Potsdam, Germany

Dennis Fischer (19)
Department of Computer Science, RWTH Aachen, Germany

Martin Josef Geiger (34)
Logistics Management Department, University of the Federal Armed Forces Hamburg, Germany

Lars Gottesbüren (27, 31)
Karlsruhe Institute of Technology, Germany

Alejandro Grez (20)
Pontificia Universidad Católica de Chile, Santiago, Chile; Millennium Institute for Foundational Research on Data, Santiago, Chile

Mario Grobler (30)
University of Bremen, Germany

Carla Groenland (9)
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Mohit Gurumukhani (3)
University of California at San Diego, CA, USA

Venkatesan Guruswami (5)
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA

Michael Hamann (27, 31)
Karlsruhe Institute of Technology, Germany

Tim A. Hartmann (19)
Department of Computer Science, RWTH Aachen, Germany

Marc Heinrich (29, 33)
University of Leeds, UK

Zacharias Heinrich (7)
Universität zu Lübeck, Germany

Emanuel Herrendorf (28)
Computer Science, Philipps Universität Marburg, Germany

Tobias Heuer (27, 31)
Karlsruhe Institute of Technology, Germany

Petr Hliněný (6)
Masaryk University, Brno, Czech Republic

Russell Impagliazzo (3)
University of California at San Diego, CA, USA

Michael Jaber (3)
University of California at San Diego, CA, USA

Hugo Jacob (21)
ENS Paris-Saclay, France

Bart M. P. Jansen (14)
Eindhoven University of Technology, The Netherlands

Vít Jelínek (22)
Computer Science Institute, Charles University, Prague, Czech Republic

Lawqueen Kanesh (23)
National University of Singapore, Singapore

Yuping Ke (13)
Department of Computing, Hong Kong Polytechnic University, Hong Kong, China

Leon Kellerhals (26)
Algorithmics and Computational Complexity, Faculty IV, Technische Universität Berlin, Germany

Eun Jung Kim (10)
Université Paris-Dauphine, PSL University, CNRS UMR7243, LAMSADE, Paris, France

Tomohiro Koana (26)
Algorithmics and Computational Complexity, Faculty IV, Technische Universität Berlin, Germany

Marvin Künnemann (3)
Institute for Theoretical Studies, ETH Zürich, Switzerland

Stefan Lendl (19)
Institute of Operations und Information Systems, Universität Graz, Austria

Shaohua Li (24)
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Jayakrishnan Madathil (23)
Chennai Mathematical Institute, Chennai, India

Bertrand Marchand (11)
LIX, CNRS UMR 7161, Ecole Polytechnique, Institut Polytechique de Paris, France; LIGM, Marne-la-Valée, France

Filip Mazowiecki (20)
Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany

André Nichterlein (26)
Algorithmics and Computational Complexity, Faculty IV, Technische Universität Berlin, Germany

Maria Paula Parga Nina (3)
University of California at San Diego, CA, USA

Michal Opler (22)
Computer Science Institute, Charles University, Prague, Czech Republic

Jakub Pekárek (22)
Department of Applied Mathematics, Charles University, Prague, Czech Republic

Anthony Perez (17)
Univ. Orléans, INSA Centre Val de Loire, LIFO EA 4022, F-45067 Orléans, France

Théo Pierron (8, 29, 33)
LIRIS, CNRS, Université Claude Bernard Lyon 1, Université de Lyon, France

Marcin Pilipczuk (21, 24)
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Michał Pilipczuk (20)
University of Warsaw, Poland

Yann Ponty (11)
LIX, CNRS UMR 7161, Ecole Polytechnique, Institut Polytechique de Paris, France

Ulysse Prieto (29, 33)
Independent Researcher, Paris, France

Gabriele Puppis (20)
University of Udine, Italy

Roman Rabinovich (30)
Technische Universität Berlin, Germany

Ashutosh Rai (18)
Department of Mathematics, IIT Delhi, India

Igor Razgon (25)
Department of Computer Science and Information Systems, Birkbeck University of London, UK

Amadeus Reinald (10)
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Rüdiger Reischuk (7)
Universität zu Lübeck, Germany

Cristian Riveros (20)
Pontificia Universidad Católica de Chile, Santiago, Chile; Millennium Institute for Foundational Research on Data, Santiago, Chile

Peter Rossmanith (12)
Department of Computer Science, RWTH Aachen, Germany

Abhishek Sahu (23)
The Institute of Mathematical Sciences, HBNI, Chennai, India

Ignasi Sau (4)
LIRMM, Université de Montpellier, CNRS, France

Saket Saurabh (1, 23)
Institute of Mathematical Sciences, Chennai, India; University of Bergen, Norway

Yannik Schnaubelt (30)
University of Bremen, Germany

Sebastian Siebertz (30)
University of Bremen, Germany

Maximilian Sonneborn (30)
University of Bremen, Germany

Jonas Spinner (27, 31)
Karlsruhe Institute of Technology, Germany

Sylwester Swat (32, 35)
Institute of Computing Science, Poznań University of Technology, Poland

Céline M. F. Swennenhuis (9)
Department of Mathematics and Computer Science, Eindhoven University of Technology, The Netherlands

Till Tantau (7)
Universität zu Lübeck, Germany

Stéphan Thomassé (10)
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Ioan Todinca (17)
Univ. Orléans, INSA Centre Val de Loire, LIFO EA 4022, F-45067 Orléans, France

Shaily Verma (23)
The Institute of Mathematical Sciences, HBNI, Chennai, India

Rémi Watrigant (10)
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Christopher Weyand (27, 31)
Karlsruhe Institute of Technology, Germany

Marcus Wilhelm (27, 31)
Karlsruhe Institute of Technology, Germany

Gerhard J. Woeginger (19)
Department of Computer Science, RWTH Aachen, Germany

Michał Włodarczyk (14)
Eindhoven University of Technology, The Netherlands

Philipp Zschoche (26)
Algorithmics and Computational Complexity, Faculty IV, Technische Universität Berlin, Germany

A Polynomial Kernel for Deletion to Ptolemaic Graphs

Akanksha Agrawal
Indian Institute of Technology Madras, Chennai, India

Aditya Anand
Indian Institute of Technology Kharagpur, India

Saket Saurabh
Institute of Mathematical Sciences, Chennai, India
University of Bergen, Norway

Abstract

For a family of graphs $\mathcal{F}$, given a graph G and an integer k, the $\mathcal{F}$-Deletion problem asks whether we can delete at most k vertices from G to obtain a graph in the family $\mathcal{F}$. The $\mathcal{F}$-Deletion problems for all non-trivial families $\mathcal{F}$ that satisfy the hereditary property on induced subgraphs are known to be NP-hard by a result of Yannakakis (STOC'78). Ptolemaic graphs are the graphs that satisfy the Ptolemy inequality, and they are the intersection of chordal graphs and distance-hereditary graphs. Equivalently, they form the set of graphs that do not contain any chordless cycles or a gem as an induced subgraph. (A gem is the graph on 5 vertices, where four vertices form an induced path, and the fifth vertex is adjacent to all the vertices of this induced path.) The Ptolemaic Deletion problem is the $\mathcal{F}$-Deletion problem, where $\mathcal{F}$ is the family of Ptolemaic graphs. In this paper we study Ptolemaic Deletion from the viewpoint of Kernelization Complexity, and obtain a kernel with $\mathcal{O}(k^6)$ vertices for the problem.

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

Keywords and phrases Ptolemaic Deletion, Kernelization, Parameterized Complexity, Gem-free chordal graphs

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.1

Related Version *Full Version*: https://akanksha-agrawal.weebly.com/uploads/1/2/2/2/122276497/ptolemaic-deletion.pdf

Funding *Akanksha Agrawal*: Supported by New Faculty Initiation Grant (IIT Madras).
Saket Saurabh: Supported by funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819416) and Swarnajayanti Fellowship grant DST/SJF/MSA-01/2017-18.

1 Introduction

Graph modification problems are one of the central problems in Graph Theory, and vertex deletion is one of the most natural graph modification operations. For a family of graphs $\mathcal{F}$, $\mathcal{F}$-Deletion takes as input an n-vertex graph G and an integer k, and the objective is to determine whether we can remove a set of at most k vertices from G to obtain a graph in $\mathcal{F}$. Some of the classical examples of $\mathcal{F}$-Deletion are the NP-hard problems like Vertex Cover, Feedback Vertex Set, and Odd Cycle Transversal, corresponding to $\mathcal{F}$ being the family of edgeless graphs, forests, and bipartite graphs, respectively. Unfortunately, most of these $\mathcal{F}$-Deletion problems are NP-hard by a result of [19, 22]. Thus, they have received substantial attention in the algorithmic paradigms for coping with NP-hardness, including Approximation Algorithms and Parameterized Complexity.

In Parameterized Complexity each problem instance is accompanied by a parameter k. One of the central notions in parameterized complexity is that of *fixed parameter tractability* (FPT). A parameterized problem Π is FPT if given an instance (I, k) of Π, we can determine whether (I, k) is a YES-instance of Π in time bounded by $\mathcal{O}(f(k)|I|^{\mathcal{O}(1)})$, where f is some computable function of k and $|I|$ is the encoding length of I. One way to obtain an FPT algorithm for a (decidable) parameterized problem algorithm is to exhibit a *kernelization algorithm*, or *kernel*. A kernel for a problem Π is an algorithm that given an instance (I, k) of Π, runs in polynomial time and outputs an equivalent instance (I', k') of Π such that $|I'|, k'$ are both upper bounded by $g(k)$ for some computable function g. The function g is the *size* of the kernel, and if g is a polynomial function, then we say that the kernel is a *polynomial kernel*. A kernel for a decidable problem implies that it admits an FPT algorithm, but kernels are also very interesting in their own right, as they mathematically capture the efficiency of polynomial time pre-processing routines.

A graph is a *chordal* graph if it does not contain any induced cycle on at least four vertices. A graph G is *distance hereditary* if the distances between vertices in every connected induced subgraph of G are the same as in the graph G. *Ptolemaic graphs* are the graphs that are both chordal and distance hereditary. In this paper we study the $\mathcal{F}$-Deletion problem corresponding to the family of Ptolemaic graphs. Formally, we study the following problem.

Ptolemaic Deletion	**Parameter:** k
Input: A graph G with n vertices and a non-negative integer k.	
Question: Is there $X \subseteq V(G)$ such that $\|X\| \leqslant k$ and $G \setminus X$ is a Ptolemaic graph?	

We study the parameterized complexity of the Ptolemaic Deletion problem. Recently, Ahn et al. [7] obtained a constant-factor approximation algorithm for the weighted version of the problem. In their algorithm, for a given instance of Ptolemaic Deletion, they create an equivalent (and approximation preserving) instance of a special Feedback Vertex Set problem, which they call Feedback Vertex Set with Precedence Constraints. With a minor modification to the well-known iterative-compression based FPT algorithm for Feedback Vertex Set (see, for example [9]), we can obtain a $c^k n^{\mathcal{O}(1)}$-time algorithm for Feedback Vertex Set with Precedence Constraints. The above together with the reduction from Ptolemaic Deletion to the problem Feedback Vertex Set with Precedence Constraints presented in [7] implies that Ptolemaic Deletion admits a simple $c^k n^{\mathcal{O}(1)}$-time FPT algorithm. We remark that both Chordal Vertex Deletion [20] and Distance Hereditary Deletion [13] are FPT, but their algorithms are more involved, compared to the elegant FPT algorithm that Ptolemaic Deletion admits.

Due to the existence of a simple single-exponential FPT algorithm for Ptolemaic Deletion, in this paper we focus on obtaining a polynomial kernel for the problem. In particular, we obtain the following result.

▶ **Theorem 1.** Ptolemaic Deletion *admits a kernel with* $\mathcal{O}(k^6)$ *vertices.*

We note that the kernelization complexity of Chordal Vertex Deletion was a well-known open problem in the field of Parameterized Complexity. Jansen and Pilipczuk [16] designed the first polynomial kernel for the above problem, and shortly afterwards, Agrawal et al. [3] gave an improved kernel with $\mathcal{O}(k^{13})$ vertices. Kim and Kwon [17] showed that Distance Hereditary Deletion admits a kernel of size $\mathcal{O}(k^{30} \log^5 k)$. We remark that the kernel we obtain has only $\mathcal{O}(k^6)$ vertices, which is much smaller than the kernels for both these problems. We also believe that our techniques can prove to be useful in obtaining/improving kernels for related classes of graphs.

Our Methods. We now describe the techniques we use in obtaining our kernel, for the given instance (G, k) of Ptolemaic Deletion. First, we compute an approximate solution S, using the result of Ahn et al. [7]. We further construct a strengthened version of the approximate solution called a redundant solution, D, introduced in [4]. Roughly speaking, redundant solutions allow us to, in a sense, forget about small obstructions. This simplifies many technical difficulties, while only maintaining a small set of vertices. We analyse the set of maximal cliques and bound the size of any maximal clique in the Ptolemaic graph $G \setminus (S \cup D)$ by $\mathcal{O}(k^3)$ by exploiting structural properties of Ptolemaic graphs and the property of "redundancy" ensured by the set D. To this end, we also use matchings in auxiliary graphs, to determine which vertices are "safe" to delete.

After bounding the sizes of maximal cliques in $G \setminus (S \cup D)$, we make use of a characterization of Ptolemaic graphs based on their inter-clique digraph [21]. In particular, Uehara and Uno [21] showed that a graph is Ptolemaic if and only if the underlying undirected graph of the inter-clique digraph of the given graph is a forest. We now use the concept of independence degree introduced in [3], and bound the independence degree of certain vertices, introducing an annotated version of the problem, similar to [3]. We then design a procedure which exploits the bounded independence degree and obtain a set $R \subseteq D$, so that the number of leaves in the undirected inter-clique digraph of the Ptolemaic graph $G \setminus (R \cup S)$ can be bounded by $\mathcal{O}(k^3)$. Every vertex of this forest, which we call bags, corresponds to a set of vertices which form a clique in $G \setminus (R \cup S)$. This, together with the fact that the size of a maximal clique is bounded in $G \setminus (R \cup S)$, gives us a bound on the number of vertices contained in degree-1 and degree $\geqslant 3$ bags in the (undirected) forest. Finally, we bound the number of vertices in degree-2 bags. In order to do this, we examine the structure of degree-2 paths (paths containing only degree 2 bags of the forest) in the inter-clique digraph. We first give several structural reduction rules and then show that we can safely replace "large" portions of the graph with smaller-sized graphs, if we maintain the size of a minimum separator in an augmented graph. Combining everything together, we obtain our kernel for Ptolemaic Deletion with $\mathcal{O}(k^6)$ vertices.

Related Works. As Ptolemaic graphs are hereditary on induced subgraphs, by the seminal result of Lewis and Yannakakis [22, 19] it follows that Ptolemaic Deletion is NP-complete. As mentioned earlier, very recently Ahn et al. [7] obtained a constant factor approximation algorithm for Ptolemaic Deletion. There have been several studies of the parameterized complexity for deletion to subclasses of chordal graphs and distance hereditary graphs. As mentioned previously, both Chordal Vertex Deletion [3, 20, 16] and Distance Hereditary Deletion [13, 17] admit FPT algorithms and polynomial kernels. Interval graphs are an important subclass of chordal graphs. Cao and Marx [8] showed that Interval Vertex Deletion admits an FPT algorithm. Recently, Agrawal et al. [4] obtained the first polynomial kernel for Interval Vertex Deletion with $\mathcal{O}(k^{1741})$ vertices. Block graphs are a subclass of Ptolemaic graphs, and Block Vertex Deletion is known to admit an FPT algorithm running in time $4^k n^{\mathcal{O}(1)}$ and a kernel with $\mathcal{O}(k^4)$ vertices [2]. Split graphs form a well-known subclass of chordal graphs, and Split Vertex Deletion is known to admit a $\mathcal{O}(1.2738^k k^{\mathcal{O}(\log k)} + n^{\mathcal{O}(1)})$-time FPT algorithm [10] and a kernel with $\mathcal{O}(k^2)$ vertices [1]. Another class of graphs which form a subclass of Ptolemaic graphs are 3-leaf power graphs. The corresponding deletion problem, 3-Leaf Power Deletion was shown to be FPT by [11, 5]. Recently, Ahn et al. [6] designed a polynomial kernel for 3-Leaf Power Deletion.

2 Preliminaries

Sets and Undirected Graphs. For $k \in \mathbb{N}$, we use $[k]$ as a shorthand for $\{1, 2, \ldots, k\}$. Given an undirected graph G, we let $V(G)$ and $E(G)$ denote its vertex-set and edge-set, respectively. We let n denote the number of vertices in a graph G, whenever the context is clear. The *open neighborhood*, or simply the *neighborhood*, of a vertex $v \in V(G)$ is defined as $N_G(v) = \{w \mid \{v, w\} \in E(G)\}$. We extend the definition of neighborhood of a vertex to a set of vertices as follows. Given a subset $U \subseteq V(G)$, $N_G(U) = \bigcup_{u \in U} N_G(u) \setminus U$. We omit subscripts when the graph G is clear from the context. The *induced subgraph* $G[U]$ is the graph with vertex-set U and edge set $\{\{u, u'\} \mid u, u' \in U, \text{ and } \{u, u'\} \in E(G)\}$. Moreover, we define $G \setminus U$ as the induced subgraph $G[V(G) \setminus U]$.

We define the distance $d(u, v)$ between two vertices $u, v \in V(G)$ as the length of the shortest path between them.

For a graph G and vertices $s, t \in V(G)$, where $s \neq t$, an *s-t separator* is a subset $U \subseteq V(G)$ such that $G \setminus U$ has no s-t path. It is well known that a minimum sized s-t separator (also called a *minimum s-t separator*) can be found in polynomial time using, for example, an algorithm for maximum flow, see for instance Chapter 7 of [18].

A *gem* is a graph on five vertices, with one vertex adjacent to each of the remaining four vertices which form an induced path.

A graph G is *chordal* if it does not contain a chordless cycle as an induced subgraph. G is *distance hereditary* if for every connected induced subgraph H of G and two distinct vertices $u, v \in V(H)$, the length of the shortest path between u and v in H is equal to the length of the shortest path between u and v in G.

Ptolemaic Graphs. A graph G is a *Ptolemaic graph* if for every four vertices u, v, w, x in the same connected component, $d(u, v)d(w, x) + d(u, x)d(v, w) \geqslant d(u, w)d(v, x)$.

▶ **Proposition 2** (Theorems 2.1,3.2 [15]). *Given a graph G, the following statements are equivalent: i) G is Ptolemaic, ii) G is both chordal and distance hereditary, and iii) G does not contain a gem or a chordless cycle as an induced subgraph.*

We call gems and chordless cycles *obstructions* and say that G contains an obstruction, if it has an obstruction as an induced subgraph.

Inter-Clique Digraphs. For a graph G, let $\mathcal{C}(G)$ be the set that contains i) all maximal cliques in G, and ii) non-empty intersections of (any number of) distinct maximal cliques in G. Consider the directed graph D_G which has a vertex v_C for each $C \in \mathcal{C}(G)$. For every $X, Y \in \mathcal{C}(G)$, where i) $X \subset Y$, and ii) there is no $W \in \mathcal{C}(G)$ such that $X \subset W$ and $W \subset Y$; we add an arc from v_Y directed towards v_X. For a directed graph T, $Und(T)$ denotes the underlying undirected graph (obtained by removing the directions associated with arcs in T). We make use of a characterization of Ptolemaic Graphs based on Inter-Clique Digraphs by Uehara and Uno[21].

▶ **Proposition 3** (Theorem 5,8 [21]). *A graph G is Ptolemaic if and only if $Und(D_G)$ is a forest. Moreover, the inter-clique digraph of a Ptolemaic graph can be computed in linear time.*

For a Ptolemaic graph G, let T_G denote its inter-clique digraph. We refer to the vertices in T_G (and $Und(T_G)$) as *bags*, to avoid confusions. A *leaf* of T_G is a leaf in $Und(T_G)$. For a bag $B \in T_G$, denote the associated set of vertices in G in the bag B by $V(B)$.

In Weighted Ptolemaic Deletion, we are given a graph G and a weight function $w : V \to \mathbb{R}^+ \cup \{0\}$, and the objective is to compute a minimum weight subset $X \subseteq V(G)$ such that $G \setminus X$ is a Ptolemaic graph.

For a graph G, we say that a set $X \subseteq V(G)$ is a *solution* for G if $G \setminus X$ is a Ptolemaic graph. By $\mathsf{opt}(G)$, we denote the size of a minimum sized solution for G. Moreover, if we are given a weight function $w : V(G) \to \mathbb{R}^+ \cup \{0\}$, then $\mathsf{opt}(G)$ denotes the weight of a minimum weight solution for G.[1] The following result is known regarding Weighted Ptolemaic Deletion.

▶ **Proposition 4** (Theorem 1.1 [7]). *There is a constant $c \in \mathbb{N}$ and a polynomial time algorithm* Approx *for* Weighted Ptolemaic Deletion*, which given a graph G and a weight function $w : V(G) \to \mathbb{R}^+ \cup \{0\}$, outputs a solution for G of weight at most $c \cdot \mathsf{opt}(G)$.*

2.1 Computing a Redundant Solution

Our kernelization algorithm will also use a "strengthened" approximate solution, which will be slightly larger in size than the solution that we obtain using the known constant-factor approximation algorithm (Proposition 4). The type of strengthening of an approximate solution that we use will be the same as the notion of *redundant solution* that was introduced in the context of Interval Vertex Deletion [4]. Intuitively speaking, a redundant solution allows us to "forget" about "small" obstructions in some of our reduction rules. This is achieved with the help of an implicit family of subsets of $V(G)$ which has a guarantee that any solution for G of size at most k is a hitting set for this family. We will now formalise the above notions.

Consider a graph G. For a family $\mathcal{W} \subseteq 2^{V(G)}$, a subset $S \subseteq V(G)$ *hits* $\mathcal{W}$ if for all $W \in \mathcal{W}$, we have $S \cap W \neq \emptyset$. We say that $\mathcal{W} \subseteq 2^{V(G)}$ is *t-necessary* if every solution for G of size at most t hits $\mathcal{W}$. Moreover, we say that an obstruction $\mathbb{O}$ is *covered by* $\mathcal{W}$ if there exists $W \in \mathcal{W}$, such that $W \subseteq V(\mathbb{O})$. Now, we are ready to formally define the notion of redundant solution (for our context).

▶ **Definition 5.** For a graph G, a family $\mathcal{W} \subseteq 2^{V(G)}$ and $t \in \mathbb{N}$, a subset $M \subseteq V(G)$ is *t-redundant with respect to $\mathcal{W}$* if for any obstruction $\mathbb{O}$ that is not covered by $\mathcal{W}$, it holds that $|M \cap V(\mathbb{O})| > t$.

Let (G, k) be an instance of Ptolemaic Deletion. We will next focus on computing a t-redundant solution for G, for an appropriately defined $\mathcal{W}$ (and t).

▶ **Lemma 6.** [♠] [2] *In polynomial time we can either correctly conclude that (G, k) is a* no*-instance, or compute a $(k+1)$-necessary family $\mathcal{W} \subseteq 2^{V(G)}$ and a set $M \subseteq V(G)$, such that $\mathcal{W} \subseteq 2^M$, M is a solution for G that is 4-redundant with respect to $\mathcal{W}$ and $|M| \in O(k^5)$.*

The utility of a redundant solution is captured in the following lemma.

▶ **Lemma 7.** [♠] *Consider an instance (G, k) of* Ptolemaic Deletion*, and D be a 4-redundant solution for G with respect to a $(k+1)$-necessary family $\mathcal{W}$, returned by Lemma 6. For $v \in V(G) \setminus D$, let X be any solution of size at most k (if it exists) for $G \setminus \{v\}$. Then X hits all obstructions of size at most 5 in G. In other words, $G \setminus X$ does not contain a gem or a chordless cycle of length less than 6.*

[1] For a subset $S \subseteq V(G)$, $w(S) = \sum_{s \in S} w(S)$.

[2] Proofs of results marked with ♠ can be found in the full version of the paper.

3 Kernel for Ptolemaic Vertex Deletion

The objective of this section is to prove Theorem 1. Let (G, k) be an instance of Ptolemaic Deletion. Our algorithm begins by computing a c-approximate solution S, for the Weighted Ptolemaic Deletion instance $(G, w^*_{n+1,\emptyset})$ using Approx (see Proposition 4).[3] If $|S| > ck$, we conclude that (G, k) is a no-instance, and return a trivial no-instance of the problem as a kernel. We now assume $|S| \leqslant ck$. Next, we use Lemma 6 to compute (in polynomial time) a $(k+1)$-necessary family $\mathcal{W}$ and a 4-redundant (with respect to $\mathcal{W}$) solution D for G of size bounded by $\mathcal{O}(k^5)$. Note that we have $\mathcal{W} \subseteq 2^D$, from the lemma.

Firstly, we bound the size of a maximal clique in $G \setminus (S \cup D)$ using the structure of the connected components upon removal of a maximal clique in a Ptolemaic graph and the properties guaranteed by the 4-redundant solution D. Intuitively, for each maximal clique C in $G \setminus S$, we do the following. Firstly we mark a few vertices in C which are neighbours of some vertices in S. Secondly, we create auxiliary graphs, with the guarantee that: i) if the maximum matching in these auxiliary graphs is "large", we find a vertex v that can be safely deleted to obtain the instance $(G \setminus \{v\}, k-1)$, and ii) otherwise, we will be able to use maximum matchings in these auxiliary graphs to mark at most $\mathcal{O}(k^3)$ many vertices in C. The marked neighbours of S in C and the vertices marked using the help of auxiliary graphs will help us capture different behaviours of C in an obstruction. Finally we argue that deleting unmarked vertices in $C \setminus D$ is safe. Formally, we prove the following lemma in Section 4.

▶ **Lemma 8.** *In polynomial time we can either conclude that (G, k) is a* no*-instance, or obtain an equivalent instance (G', k'), such that $k' \leqslant k$, G' is an induced subgraph of G, such that $S \cup D \subseteq V(G')$, and each (maximal) clique in $G' \setminus (S \cup D)$ has at most $\mathcal{O}(k^3)$ vertices.*

If at any point in our algorithm, we are able to conclude that the given instance is a no-instance, then we output some trivial no-instance as a kernel with $\mathcal{O}(1)$ vertices. We use Lemma 8 for the given instance, and without loss of generality assume that we obtained an equivalent instance of Ptolemaic Deletion, which satisfies all the conditions guaranteed by the lemma. For the sake of notational simplicity, we use (G, k) as the instance of Ptolemaic Deletion that we currently have, i.e., the one returned by the lemma.

Next, we use the fact that the undirected version of the inter-clique digraph of a Ptolemaic graph is an undirected forest. We construct a suitable Ptolemaic graph $G \setminus (S \cup R)$ for a carefully constructed set $R \subseteq D$ and reduce the number of leaves in its undirected inter-clique digraph. Towards this, we use the idea of bounding the independence degree of vertices from [3]. We introduce two sets of pairs of vertices E_M and E_I, which will stand for mandatory and irrelevant pairs of vertices. Moreover, we call an edge $\{u, v\} \in E(G)$ *mandatory* if $\{u, v\} \in E_M$ and *irrelevant* if $\{u, v\} \in E_I$. An edge is *relevant* if it is not irrelevant.

For a graph H and sets $E_M, E_I \subseteq V(H) \times V(H)$, for a vertex $v \in V(H)$, by $N^H_{\mathsf{rel}}(v)$ we denote the set of vertices in the neighbourhood of v in H which are adjacent to v using an edge in $E(H) \setminus E_I$. We omit the superscript H from the above notation, whenever the context is clear.

▶ **Definition 9.** *Given a set $E_I \subseteq V(G) \times V(G)$ and a vertex $v \in V(G)$, the independence degree of v in G, denoted by $d^G_I(v)$, is the size of a maximum independent set in the graph $G[N_{\mathsf{rel}}(v)]$.*

[3] The choice of $n+1$ is arbitrary.

We remark that we omit the superscript in $d_I^G(v)$ when the graph is clear from the context. Next, we consider an annotated version of Ptolemaic Deletion similar to [3].

Augmented Ptolemaic Deletion	**Parameter:** k
Input: An undirected graph G, sets $E_M, E_I \subseteq V(G) \times V(G)$ and a non-negative integer k, such that any $X \subseteq V(G)$ which hits all chordless cycles in G which contain no edge from E_I, hits all chordless cycles in G.	
Question: Does there exist a subset $X' \subseteq V(G)$ (called a *solution* for (G, k, E_M, E_I)), such that (i) $\lvert X'\rvert \leqslant k$ (ii) For each $\{u, v\} \in E_M$, $X' \cap \{u, v\} \neq \emptyset$ and (iii) $G \setminus X'$ is a Ptolemaic graph?	

We next obtain an instance of Augmented Ptolemaic Deletion, where the number of leaves in the inter-clique digraph of the Ptolemaic graph obtained after removing S and a suitable subset of D is bounded by $\mathcal{O}(k^3)$.

▶ **Lemma 10.** [♠] *In polynomial time, either we can conclude that (G, k) is a* no*-instance, or output an instance (G', k', E'_M, E'_I) of* Augmented Ptolemaic Deletion*, a set $R \subseteq D$ and the inter-clique digraph $T_{G' \setminus (R \cup S)}$ of $G' \setminus (R \cup S)$ so that the following hold:*

1. *$k' \leqslant k$, $S, D \subseteq V(G')$, and G' is an induced subgraph of G.*
2. *for each $\{u, v\} \in E'_M$, either $u \in S$ or $v \in S$.*
3. *Each chordless cycle in G' which contains a vertex from R contains an edge from E'_I.*
4. *(G, k) is a* yes*-instance of* Ptolemaic Deletion *if and only if (G', k', E'_M, E'_I) is a* yes*-instance of* Augmented Ptolemaic Deletion*.*
5. *The neighbourhood of every vertex $r \in R$ in $G \setminus (R \cup S)$ is a clique.*
6. *The number of leaves in $T_{G' \setminus (R \cup S)}$ is bounded by $\mathcal{O}(k^3)$.*
7. *The independence degree (in G') of every vertex $s \in S$ is bounded by $\mathcal{O}(k^2)$ and $\lvert E'_M\rvert \in \mathcal{O}(k^2)$.*

We use Lemma 10, and assume that we have an instance (G', k', E'_M, E'_I) of Augmented Ptolemaic Deletion and a set $R \subseteq D$, satisfying the properties guaranteed by the lemma. Using the next lemma we obtain an instance of Augmented Ptolemaic Deletion equivalent to that of (G', k', E'_M, E'_I), with $\mathcal{O}(k^6)$ vertices.

▶ **Lemma 11.** [♠] *For the* Ptolemaic Deletion *instance (G, k), let (G', k', E'_M, E'_I) be the instance of* Augmented Ptolemaic Deletion *and $R \subseteq D$ be the set returned by Lemma 10.*[4] *In polynomial time we either conclude that (G', k', E'_M, E'_I) is a* no*-instance or output an equivalent instance $(\widetilde{G}, \widetilde{k}, \widetilde{E}_M, \widetilde{E}_I)$ of* Augmented Ptolemaic Deletion*, such that $\widetilde{k} \leqslant k'$, $\lvert V(\widetilde{G})\rvert \in \mathcal{O}(k^6)$, and $\lvert \widetilde{E}_M\rvert \leqslant \lvert E'_M\rvert$.*

To prove the above lemma, we reduce the number of vertices in 2-degree bags in the undirected inter-clique digraph of $G' \setminus (R \cup S)$. We do this by exploiting the structure of 2-degree paths, which are paths in $T_{G' \setminus (R \cup S)}$ where each bag is of degree 2. On a high level, we mark many bags in $T_{G' \setminus (R \cup S)}$ so that each 2-degree path splits into several maximal unmarked sub-paths. We ensure that the vertices of G' in bags of these subpaths do not "interact" with $R \cup S$, that is, they do not have any neighbours in $R \cup S$, using several reduction rules. We then show that we can reduce the number of vertices in these bags, by designing a reduction rule which preserves the size of a minimum separator in an augmented graph. Since we already have a bound on the number of leaves from Lemma 11, we get a

[4] We assume that in G, each maximal clique has at most $\mathcal{O}(k^3)$ vertices, using Lemma 8.

bound on the number of degree $\geqslant 3$ bags as well. Thus, proving a bound on the number of vertices present in bags of degree 2 gives us a bound on the total number of vertices in the inter-clique digraph of $G' \setminus (R \cup S)$, since we already have a bound on clique size (and hence, on the number of vertices in any bag) from Lemma 8. Accounting for each step we obtain an Augmented Ptolemaic Deletion instance with $\mathcal{O}(k^6)$ vertices.

Equipped with Lemmas 8 to 11, we are now ready to prove Theorem 1.

Proof of Theorem 1. Consider an instance $(\widehat{G}, \widehat{k})$ of Ptolemaic Deletion. If Lemma 8 returns that $(\widehat{G}, \widehat{k})$ is a no-instance of Ptolemaic Deletion, then we return some trivial no-instance with $\mathcal{O}(1)$ vertices as a kernel for Ptolemaic Deletion. Otherwise, we have an equivalent instance (G, k), where each (maximal) clique in $G \setminus (S \cup D)$ has at most $\mathcal{O}(k^3)$ vertices. Next we apply Lemma 10 for the instance (G, k). Again, if the lemma returns that (G, k) is a no-instance, we return a trivial $\mathcal{O}(1)$-vertex kernel. Otherwise, we have an Augmented Ptolemaic Deletion instance (G', k', E'_M, E'_I) and the set $R \subseteq D$, returned by the lemma. We next apply Lemma 11. Firstly consider the case when the lemma returns that (G', k', E'_M, E'_I) is a no-instance of Augmented Ptolemaic Deletion. Then together with Lemma 10, we have that (G, k), and hence, $(\widehat{G}, \widehat{k})$ is a no-instance of Ptolemaic Deletion. Now consider the case when Lemma 11 returns an instance $(\widetilde{G}, \widetilde{k}, \widetilde{E}_M, \widetilde{E}_I)$ of Augmented Ptolemaic Deletion, where $|V(\widetilde{G})| \in \mathcal{O}(k^6)$. From Lemmas 8 to 11, we obtain that $(\widehat{G}, \widehat{k})$ is a yes-instance of Ptolemaic Deletion if and only if $(\widetilde{G}, \widetilde{k}, \widetilde{E}_M, \widetilde{E}_I)$ is a yes-instance of Augmented Ptolemaic Deletion.

We next obtain an instance (G', k') of Ptolemaic Deletion such that $|V(G')| \in \mathcal{O}(k^6)$ and (G', k') is a yes-instance of Ptolemaic Deletion if and only if $(\widetilde{G}, \widetilde{k}, \widetilde{E}_M, \widetilde{E}_I)$ is a yes-instance of Augmented Ptolemaic Deletion. Initialize $G' = \widetilde{G}$ and set $k' = \widetilde{k}$. For each $\{u, v\} \in \widetilde{E}_M$, add $k + 2$ vertex disjoint paths between u and v using 2 new vertices for each path. As $|\widetilde{E}_M| \leqslant |E'_M| \in \mathcal{O}(k^2)$, we obtain that $|V(G')| \in \mathcal{O}(k^6)$. Notice that no gem in G' can contain a vertex from $V(G') \setminus V(\widetilde{G})$. For every $\{u, v\} \in \widetilde{E}_M$ and any solution X to the Ptolemaic Deletion instance (G', k') , we must have either $u \in X$ or $v \in X$, for if not, then $G' \setminus X$ must contains a chordless cycle through u, v by construction. It follows that (G', k') is a yes-instance of Ptolemaic Deletion if and only if $(\widetilde{G}, \widetilde{k}, \widetilde{E}_M, \widetilde{E}_I)$ is a yes-instance of Augmented Ptolemaic Deletion. Hence, it must be the case that (G', k') and $(\widehat{G}, \widehat{k})$ are equivalent instances of Ptolemaic Deletion, where $|V(G')| \in \mathcal{O}(k^6)$ and $k' \leqslant k$. This concludes the proof. ◀

Since the proof of each of the three main lemmas is quite involved, in this short version we defer the proofs of Lemma 10 and Lemma 11 to the full version in the interest of space. At the same time we try to present all the results towards proving Lemma 8, to the extent possible.

4 Bounding Sizes of Maximal Cliques in $G \setminus (S \cup D)$

The objective of this section is to prove Lemma 8. As S is a solution for G, $G \setminus S$ does not have any chordless cycles. We next state a well-known result regarding enumerating maximal cliques in chordal graphs.

▶ **Proposition 12** (see, for example, Theorem 4.8 [14]). *In polynomial time we can compute the set of all maximal cliques in a given chordal graph.*

From the above proposition, the number of distinct cliques and the time required to enumerate them, are both bounded by $n^{\mathcal{O}(1)}$. Thus, for the remainder of this section, we work with a fixed maximal clique C in $G \setminus S$, and our objective will be to either conclude

that the number of vertices in $C \setminus D$ is bounded by $\mathcal{O}(k^3)$, or find a vertex in C which we can safely delete from G. To achieve this, we will design some marking schemes, and show that if there are unmarked vertices, then we can safely delete them. We start with a useful lemma regarding the structure of connected components upon removal of a maximal clique from a Ptolemaic graph.

▶ **Lemma 13.** *Let $\widehat{G}$ be a Ptolemaic graph and C be a maximal clique in $\widehat{G}$. Let A be a connected component of $\widehat{G} \setminus C$. Then for every vertex $v \in V(A)$ we must have $N_{\widehat{G}}(v) \cap C = \emptyset$ or $N_{\widehat{G}}(v) \cap C = N_{\widehat{G}}(V(A)) \cap C$.*

Proof. Consider a connected component A of $\widehat{G} \setminus C$. Towards a contradiction suppose that the result is not true, and let $u, v \in V(A)$ be two distinct vertices such that $N_{\widehat{G}}(u) \cap C, N_{\widehat{G}}(v) \cap C \neq \emptyset$ and $N_{\widehat{G}}(u) \cap C \neq N_{\widehat{G}}(v) \cap C$. Furthermore, let u and v be a pair of vertices satisfying the above property that have shortest possible distance between them in A. Note that $C \not\subseteq N_{\widehat{G}}(u)$ and $C \not\subseteq N_{\widehat{G}}(v)$, as C is a maximal clique in $\widehat{G}$.

Suppose that $\{u, v\} \in E(\widehat{G})$. If there exist vertices $a, b \in V(C)$ such that $a \in N_{\widehat{G}}(u) \setminus N_{\widehat{G}}(v)$, $b \in N_{\widehat{G}}(v) \setminus N_{\widehat{G}}(u)$ then $\widehat{G}[\{u, v, a, b\}]$ is a chordless cycle in $\widehat{G}$, which is a contradiction, as $\widehat{G}$ is Ptolemaic graph. Therefore, either $N_{\widehat{G}}(u) \cap C \subseteq N_{\widehat{G}}(v) \cap C$ or $N_{\widehat{G}}(v) \cap C \subseteq N_{\widehat{G}}(u) \cap C$. Assume without loss of generality that $N_{\widehat{G}}(u) \cap C \subseteq N_{\widehat{G}}(v) \cap C$. This together with the choice of u and v implies that there are vertices $a \in (N_{\widehat{G}}(v) \cap C) \setminus N_{\widehat{G}}(u)$, $b \in N_{\widehat{G}}(u) \cap C$ and $x \in C \setminus (N_{\widehat{G}}(u) \cup N_{\widehat{G}}(v))$. But then, $\widehat{G}[\{a, b, x, u, v\}]$ is a gem in $\widehat{G}$, which is a contradiction.

Now suppose $\{u, v\} \notin E(\widehat{G})$ and consider the shortest path P between u, v in A. Note that P has at least one more vertex apart from u and v, as $\{u, v\} \notin E(\widehat{G})$. Firstly consider the case when there are vertices $a \in (N_{\widehat{G}}(u) \cap C) \setminus N_{\widehat{G}}(v)$, $b \in (N_{\widehat{G}}(v) \cap C) \setminus N_{\widehat{G}}(u)$. Note that by the choice of u and v (of them being at shortest distance in A satisfying the assumed properties), no internal vertex of P can be adjacent to a or b. But then P along with the edges $\{a, b\}, \{a, u\}$ and $\{v, b\}$ forms a chordless cycle in $\widehat{G}$, which is a contradiction. We now consider the case when there are no such a and b, as assumed previously. This implies that either $N_{\widehat{G}}(u) \cap C \subseteq N_{\widehat{G}}(v) \cap C$ or $N_{\widehat{G}}(v) \cap C \subseteq N_{\widehat{G}}(u) \cap C$. Suppose that $N_{\widehat{G}}(u) \cap C \subseteq N_{\widehat{G}}(v) \cap C$ (the other case is symmetric). Now consider a vertex $a \in N_{\widehat{G}}(v) \cap N_{\widehat{G}}(u) \cap C$. If there is an internal vertex of P that is not adjacent to a, then we conclude that $\widehat{G}[V(P) \cup \{a\}]$ contains a chordless cycle. If every internal vertex of P is adjacent to a, P must have exactly 1 vertex, say w, apart from $\{u, v\}$, since otherwise we would obtain a gem obstruction in $\widehat{G}[V(P) \cup \{a\}]$. Since $N_{\widehat{G}}(u) \cap C \neq N_{\widehat{G}}(v) \cap C$ we must have either $N_{\widehat{G}}(w) \cap C \neq N_{\widehat{G}}(u) \cap C$ or $N_{\widehat{G}}(w) \cap C \neq N_{\widehat{G}}(v) \cap C$. In either case, since $\{u, w\}, \{w, v\} \in E(\widehat{G})$, this is impossible from the analysis in the preceeding paragraph of the case when $\{u, v\} \in E(\widehat{G})$. It follows that such an a cannot exist, and therefore $N_{\widehat{G}}(v) \cap N_{\widehat{G}}(u) \cap C = \emptyset$. Together, we must have either $N_{\widehat{G}}(v) \cap C = \emptyset$ or $N_{\widehat{G}}(u) \cap C = \emptyset$, a contradiction. ◀

Recall that we computed D, a 4-redundant solution with respect to $\mathcal{W}$. This will allow us to only focus on chordless cycles of length at least 6 (see Lemma 7). A chordless cycle can contain at most 2 vertices from C, as C is a (maximal) clique. If C has many vertices, we would like to argue that, we can find a vertex $c \in C \setminus D$ to delete from G. Roughly speaking, to achieve this, for every pair of vertices u, v, that are neighbours of c in some chordless cycle, we would like to ensure that, after removing some solution of size at most k, there is either (i) at least one marked common neighbour $c' \in C$ of u and v or (ii) two distinct marked vertices $c_1, c_2 \in C$, such that $\{u, c_1\}, \{v, c_2\} \in E(G)$. We show, through a simple observation, that preserving such marked vertices is enough to safely delete the vertex c from the input instance (G, k) to obtain the instance $(G \setminus \{c\}, k)$.

▶ **Observation 14.** *Consider a graph $\widehat{G}$ and a spanning cycle K of $\widehat{G}$. Then the graph $\widehat{G}$ contains an obstruction if one of the following holds: i) $|V(\widehat{G})| \geqslant 5$ and there is $v \in V(K)$, such that for each $\{x, y\} \in E(\widehat{G}) \setminus E(K)$, $v \in \{x, y\}$, or ii) $|V(\widehat{G})| \geqslant 7$ and there is $\{u, v\} \in E(K)$, such that for each $\{x, y\} \in E(\widehat{G}) \setminus E(K)$, we have $\{u, v\} \cap \{x, y\} \neq \emptyset$.*

Proof. To prove (i), notice that if v is not adjacent (in $\widehat{G}$) to at least one vertex in $V(K) \setminus \{v\}$, then $\widehat{G}$ must contain a chordless cycle. Otherwise, since K has at least 5 vertices, any four consecutive vertices a, b, c, d on K, all different from v, together with v give us a gem.

To prove (ii), let us consider five consecutive vertices of the cycle K starting from u, v, taken in order, say u, v, a, b, c. We claim that apart from its two neighbours u and a in K, the vertex v can be adjacent to only the vertices b and c in $\widehat{G}$. Suppose this is not the case. Then we get a cycle K' with $V(K') \subseteq V(K)$ in $\widehat{G}$ of length at least 5 which does not include u. Applying part (i) with the graph $\widehat{G}[V(K')]$ then gives an obstruction in $\widehat{G}$.

Therefore v can only be adjacent to u, a, b, c in K. Consider the case when v is adjacent to b but not c. Let K' be the cycle in $\widehat{G}$, obtained from K by deleting a and adding the edge $\{v, b\}$. Notice that $\widehat{G}[V(K')]$ and the spanning cycle K' of $\widehat{G}[V(K')]$, satisfy part (i) of the observation. Thus, $\widehat{G}[V(K')]$ (and hence $\widehat{G}$) must contain an obstruction. For the case when v is adjacent to c, we can obtain an obstruction in $\widehat{G}$ by using arguments similar to our previous case. Finally, if v is not adjacent to both b and c, then $\widehat{G}$ and K satisfy the conditions of (i). ◀

Now we define the notions of replacement vertices.

▶ **Definition 15.** Given a vertex $c \in C$, a set $X \subseteq V(G) \setminus \{c\}$ of size at most k, and a chordless cycle K of length at least 6 in $G \setminus X$ containing c, a vertex $c' \in C \setminus (X \cup V(K))$ is a *replacement* for c in K, if c' is adjacent to both the neighbours of c in K.

▶ **Definition 16.** Given a vertex $c \in C$, a set $X \subseteq V(G) \setminus \{c\}$ of size at most k, and a chordless cycle K of length at least 6 in $G \setminus X$ containing c, a pair of distinct vertices $c_1, c_2 \in C \setminus (X \cup V(K))$, is a *replacement* for c in K, if $\{u, c_1\}, \{v, c_2\} \in E(G)$, where u, v are the neighbours of c in K.

The purpose of this notion of replacements for a vertex is captured in the following lemma whose proof follows from Observation 14.

▶ **Lemma 17.** *Consider any vertex $c \in C$. Let $X \subseteq V(G)$ be a set of vertices such that $c \notin X$. Suppose that $G \setminus X$ contains a chordless cycle K of length at least 6 which includes the vertex c. Now suppose that there is either a single replacement vertex c' or a pair of replacement vertices c_1, c_2 for c in K, such that $c' \in C \setminus (X \cup V(K))$ and $c_1, c_2 \in C \setminus (X \cup V(K))$, as the case may be. Then $G \setminus (X \cup \{c\})$ must contain an obstruction.*

Proof. Let u and v be the neighbours of c in K. If we have a single replacement vertex c' for c in K, then consider the cycle K' obtained by replacing the path ucv by the path $uc'v$ in K. The cycle K' is clearly of length at least 6. Applying part (i) of Observation 14 to the graph $G[V(K')]$, we must have an obstruction in $G \setminus (X \cup \{c\})$. Likewise, if we have a pair of distinct vertices $c_1, c_2 \in C$ as a replacement for c in K, then consider the cycle K' obtained by replacing the path ucv by the path uc_1c_2v. The cycle K' is of length at least 7, therefore by item (ii) of Observation 14 applied to the graph $G[V(K')]$, $G \setminus (X \cup \{c\})$ must contain an obstruction. ◀

Now suppose that we mark a certain number of vertices in C. Consider any unmarked vertex $c \in C \setminus D$, and let X be a solution of size at most k for $G \setminus \{c\}$. If X is not a solution for G, then $G \setminus X$ must contain an obstruction that contains the vertex c. This

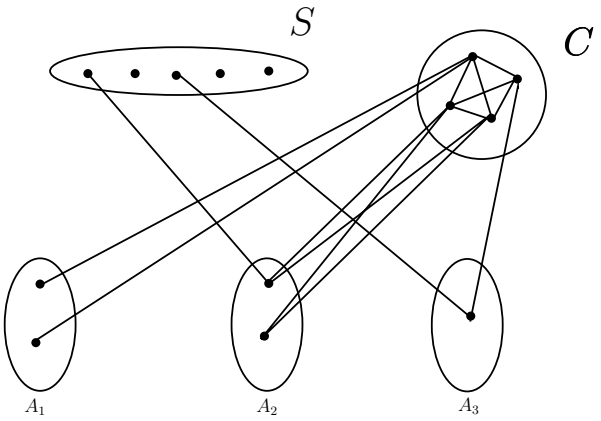


Figure 1 Bounding the size of a maximal clique C. The figure shows the components of $G \setminus (S \cup C)$ for a particular instance. For every component A and vertex $v_A \in A$, $N(v_A) \cap C = \emptyset$ or $N(v_A) \cap C = N(A) \cap C$.

obstruction must be a chordless cycle K of length at least 6 since $c \notin D$. If we design our marking scheme in such a way that for every such obstruction K, there is either (i) a marked replacement vertex c' such that $c' \notin (X \cup V(K))$ or (ii) a pair of distinct marked replacement vertices c_1, c_2 for c in K such that $c_1, c_2 \notin (X \cup V(K))$ as per the definitions above, then by Lemma 17, $G \setminus (X \cup \{c\})$ must contain an obstruction - a contradiction since X was a solution for $G \setminus \{c\}$. It follows that X must be a solution for G, and therefore the instances (G, k) and $(G \setminus \{c\}, k)$ are equivalent. Thus, given that we design our marking scheme to satisfy the above, we may delete an unmarked vertex from the input instance G without reducing the parameter.

We now describe our marking scheme and formalize the ideas from above, by looking at where the neighbours of an unmarked vertex in a chordless cycle can potentially lie. Whenever we say - mark any ℓ vertices satisfying a property P, if there are less than ℓ vertices satisfying P, then we mark all vertices which satisfy P. We start by designing our marking scheme to handle the case when the neighbours of an unmarked vertex in a chordless cycle lie in S.

▶ **Marking Scheme 1.** *For every pair of (not necessarily distinct) vertices $s_1, s_2 \in S$, mark any $k+2$ vertices in $N(s_1) \cap N(s_2) \cap C$.*

To consider the cases when the neighbours of a potential unmarked vertex lie in connected components of $G \setminus (S \cup C)$, we make use of two auxiliary bipartite graphs H_s^1 and H_s^2, corresponding to each vertex s in S. Let us denote the set of connected components of $G \setminus (S \cup C)$ by $\mathcal{A}$. The vertex set of H_s^1 is $V_s^1 \cup V_2$ where $V_s^1 = \{c \in C \ : (s, c) \notin E(G)\}$ and $V_2 = \{v_A \ : A \in \mathcal{A}\}$. We add an edge from a vertex $c \in V_s^1$ and a vertex $v_A \in V_2$ whenever both c and s have (possibly different) neighbours in A. The vertex set of H_s^2 is $V_s^2 \cup V_2$, where $V_s^2 = \{c \in C \ : (s, c) \in E(G)\}$. We add an edge from a vertex $c \in V_s^2$ and a vertex $v_A \in V_2$ whenever there is an obstruction in the subgraph induced on $\{s, c\} \cup V(A)$. We have the following observation that follows from the above description and Proposition 3.[5]

▶ **Observation 18.** *For $s \in S$, H_s^1 and H_s^2 can be constructed in polynomial time.*

[5] We note that there are other well-known polynomial time recognition algorithm for Ptolemaic graphs, that can also be used to infer that H_s^2, for $s \in S$, can be constructed in polynomial time.

Roughly speaking, we next show that, for $s \in S$ and two edges of H_s^1, incident to different vertices in V_2, we can construct an obstruction in G.

▶ **Lemma 19.** *Consider $s \in S$ and edges $\{c_1, v_{A_1}\}, \{c_2, v_{A_2}\} \in E(H_s^1)$, where $c_1, c_2 \in V_s^1$ and $A_1, A_2 \in \mathcal{A}$, where $A_1 \neq A_2$. Then, $G[V(A_1) \cup V(A_2) \cup \{s, c_1, c_2\}]$ has a chordless cycle.*

Proof. By the construction of V_s^1, $\{s, c_1\}, \{s, c_2\} \notin E(G)$. For $i, j \in \{1, 2\}$, let P_{ij} (if it exists) be a shortest path between s and c_i such that every internal vertex is from A_j. By the construction of H_s^1, P_{11} and P_{22} must necessarily exist. Notice that if at least one of c_1 or c_2, say c_1, is a neighbour of both v_{A_1} and v_{A_2} in H_s^1, then $G[\{s, c_1\} \cup V(P_{11}) \cup V(P_{12})]$ is a chordless cycle. Otherwise, the vertices s, c_1, c_2 together with the paths P_{11}, P_{22} and the edge $\{c_1, c_2\}$ constitute a chordless cycle $G[\{s, c_1, c_2\} \cup V(P_{11}) \cup V(P_{22})]$ in $G[V(A_1) \cup V(A_2) \cup \{s, c_1, c_2\}]$. ◀

For $s \in S$, we now attempt to find $(k+2)$-sized matchings in the graphs H_s^1 and H_s^2. Note that the existence of a $(k+2)$-sized matching in H_{s1} would imply that (at least) two matching edges "remain", even after we delete at most k vertices from G. The above together with Lemma 19 will allow us to conclude that s must belong to every solution for G of size at most k. The definition of H_s^2 implies that even a single "remaining" matching edge in H_s^2 would give us an obstruction, if s is not included in the solution.

▶ **Lemma 20.** *Consider $s \in S$, and let M_s^1 and M_s^2 be maximum matchings in H_s^1 and H_s^2, respectively. If either $|M_s^1| \geqslant k+2$ or $|M_s^2| \geqslant k+2$, then any solution for G of size at most k must include the vertex s.*

Proof. Towards a contradiction, suppose that X is a solution for G of size at most k which does not include the vertex s. Consider the case when $|M_s^1| \geqslant k+2$. It follows that there are distinct components $A_1, A_2 \in \mathcal{A}$, and vertices $c_1, c_2 \in V_s^1$, so that (i) $X \cap (V(A_1) \cup V(A_2) \cup \{c_1, c_2\}) = \emptyset$ and (ii) $\{c_1, v_{A_1}\}, \{c_2, v_{A_2}\} \in E(H_s^1)$ hold. But then Lemma 19 implies that $G \setminus X$ contains an obstruction, which is a contradiction. Next consider the case when $|M_s^2| \geqslant k+2$. Again as X is of size at most k, we can obtain that there is a component $A \in \mathcal{A}$ and a vertex $c \in V_s^2$ so that (i) $X \cap (V(A) \cup \{c\}) = \emptyset$ and (ii) $\{c, v_A\} \in E(H_s^2)$ hold. By the definition of H_s^2, this implies that $G \setminus X$ contains an obstruction in the subgraph induced on $V(A) \cup \{s, c\}$, which is a contradiction. ◀

Using the above lemma, we devise our first reduction rule.

▶ **Reduction Rule 1.** *If there is $s \in S$, such that $|M_s^1| \geqslant k+2$ or $|M_s^2| \geqslant k+2$, then return $(G \setminus \{s\}, k-1)$.*

The correctness of the above reduction rule follows from Lemma 20, and the polynomial time applicability of it follows from the fact that maximum matching in a graph can be computed in polynomial time (see, for example [12]). Hereafter we assume that Reduction rule 1 is not applicable, and we proceed with our next marking scheme.

▶ **Marking Scheme 2.** *For each $s \in S$, let M_s^1 and M_s^2 be maximum matchings in H_s^1 H_s^2, respectively. Then we do the following: i) for each $i \in \{1, 2\}$, we mark all the vertices in $V(M_s^i) \cap C$, ii) for each $A \in \mathcal{A}$, where $v_A \in V(M_s^1)$, mark any $k+2$ vertices in $N(V(A)) \cap V_s^1$, and iii) for each $A \in \mathcal{A}$, where $v_A \in V(M_s^2)$, mark any $k+2$ vertices in $N(V(A)) \cap V_s^2$.*

The next reduction rule deletes unmarked vertices in C.

▶ **Reduction Rule 2.** *If there is $c \in C \setminus D$ that is not marked by Marking Scheme 1 or 2, then return $(G \setminus \{c\}, k)$.*

As Marking Scheme 1 and 2 can be executed in polynomial time, the above reduction rule can be executed in polynomial time as well. In the next lemma we show its safeness.

▶ **Lemma 21.** *Reduction rule 2 is safe.*

Proof. Consider $c \in C \setminus D$ that is not marked by Marking Scheme 1 or 2. As Ptolemaic graphs are closed under induced subgraphs, to prove the lemma, it is enough to argue that a solution X for $G \setminus \{c\}$ of size at most k is also a solution for G. Towards a contradiction suppose that X is not a solution for G. Thus, $G \setminus X$ has an obstruction K, containing the vertex c. Since $c \notin D$, by Lemma 7, we may assume that K is a chordless cycle of length at least 6. Next we will construct an obstruction in $(G \setminus \{c\}) \setminus X$, and thus contradict that X is a solution for $G \setminus \{c\}$.

Firstly, consider the case when K has no other vertex from C, apart from c. Let u and v be the neighbours of c in K. Lemma 17 implies that, if we manage to find a replacement for c from $C \setminus (X \cup V(K))$ (see Definition 15 and 16), then we can find an obstruction in $(G \setminus \{c\}) \setminus X$. Firstly consider the case when $u, v \in S$. Since $k+2$ common neighbours of u, v in C were marked by Marking Scheme 1, there is at least one common neighbour $c' \in (V(G) \setminus (X \cup \{c\})) \cap C$ which forms a replacement for c. This together with Lemma 17 and the premise of the case that K has no other vertex apart from c from C implies that $(G \setminus \{c\}) \setminus X$ contain an obstruction.

Recall that we are in the case when K contain exactly one vertex, namely, c from C. Next we suppose that $u \in S$ and $v \in V(A)$, for some $A \in \mathcal{A}$. Let s be the first vertex from S on the path from v to u in $K \setminus \{c\}$ (Clearly, such an s must exist since S is an approximate solution). Notice that every internal vertex on the path from c to s, containing v, on K must be from A. Consider the case when $u \neq s$. As K is a chordless cycle, we can obtain that $\{s, c\} \notin E(G)$, and thus $\{c, v_A\} \in E(H^1_s)$. Since c is unmarked, $c \notin V(M^1_s)$. Moreover, as M^1_s is a maximum matching in H^1_s, it follows that $v_A \in V(M^1_s)$. Recall that Marking Scheme 2 marked $k+2$ vertices from $N(V(A)) \cap V^1_s$. Thus, there is a vertex $c' \in (C \setminus (X \cup \{c\})) \cap N(V(A)) \cap V^1_s$. By Lemma 13, since $\{v, c\} \in E(G)$ we must have $\{v, c'\} \in E(G)$. Marking Scheme 1 marked $k+2$ neighbors of $u \in S$ from C. Thus, there must exist a vertex $c_u \in C \setminus (X \cup \{c\})$, such that $\{c_u, u\} \in E(G)$. If $c' = c_u$, the vertex c' is a replacement for c with respect to K, and otherwise, the pair of vertices c', c_u forms a replacement for c in K. In either case, using Lemma 17 and the fact that K is a chordless cycle on at least 6 vertices, we can obtain an obstruction in $(G \setminus \{c\}) \setminus X$, for the case when $u \neq s$. Next we consider the case when $u = s$. Notice that $\{c, v_A\}$ is an edge in H^2_s. As c is unmarked, it must be the case that $v_A \in V(M^2_s)$. Since $k+2$ vertices in $N(V(A)) \cap C \cap N(s)$ were marked by Marking Scheme 2, there is a vertex $c' \in (N(V(A)) \cap N(s) \cap C) \setminus (X \cup \{c\})$. Again, by Lemma 13 we obtain that $\{v, c'\} \in E(G)$, and therefore the vertex c' forms a replacement for c, and hence from Lemma 17 we can construct an obstruction in $(G \setminus \{c\}) \setminus X$.

Next we consider the case when $u \in V(A_1)$ and $v \in V(A_2)$, for some (not necessarily distinct) $A_1, A_2 \in \mathcal{A}$, and K has no other vertex from C, apart from c. Let s_1, s_2 be the first vertices from S on the paths from u to v and v to u in $K \setminus \{c\}$, respectively. Observe that $\{c, v_{A_1}\} \in E(H^1_{s_1})$ and $\{c, v_{A_2}\} \in E(H^1_{s_2})$. Since c is unmarked, we must have $v_{A_1} \in M^1_{s_1}$ and $v_{A_2} \in M^1_{s_2}$. Marking Scheme 2 ensures that there are c'_1, $c'_2 \in C \cap V(G) \setminus (X \cup \{c\})$, where $c'_1 \in N(V(A_1))$ and $c'_2 \in N(V(A_2))$ which, by Lemma 13, are adjacent to vertices u, v respectively. If $c'_1 = c'_2$, we use c'_1 as a replacement for c in K, else the pair c'_1, c'_2 forms a replacement for c.

Now we consider the case when the chordless cycle K contains two vertices from C. (Note that since K is a chordless cycle, it can contain at most 2 vertices from C.) Since K contains c, exactly one of $u, v \notin C$, say $u \notin C$ (the other case can be argued symmetrically). We

further consider the following cases based on whether $u \in S$. Firstly consider the case when $u \in S$. Since Marking Scheme 1 marked $k+2$ neighbours of $u \in S$ from C, it follows that there is a vertex $c' \in C \setminus (X \cup V(K))$ which is a replacement for c, and thus using Lemma 17 we can obtain an obstruction in $(G \setminus \{c\}) \setminus X$. Now suppose that $u \in V(A)$ for some $A \in \mathcal{A}$. Let s be the first vertex from S on the path from u to v in $K \setminus \{c\}$. This implies that $\{c, v_A\} \in E(H_s^1)$ and that $v_A \in V(M_s^1)$. Therefore, $k+2$ vertices in $N((V(A)) \cap V_s^1$ must have been marked by Marking Scheme 2. Since $|X| \leqslant k$, c is an unmarked vertex, and $|C \cap V(K)| \leqslant 2$, there must exist a marked vertex $c' \in (N(V(A)) \cap V_s^1) \setminus (X \cup V(K))$. By Lemma 13 we must have $\{u, c'\} \in E(G)$. Thus, c' forms a replacement for c in K, and we obtain an obstruction in $(G \setminus \{c\}) \setminus X$, using Lemma 17. ◀

Proof of Lemma 8. For the given instance (G, k) of Ptolemaic Deletion, an approximate solution S for G of size $\mathcal{O}(k)$, and a solution D for G that is 4-redundant with respect to a $(k+1)$-necessary family $\mathcal{W}$, returned by Lemma 6, we do the following. In polynomial time, we enumerate the set $\mathcal{C}$, of all maximal maximal cliques in $G \setminus S$ using Observation 12. For each clique $C \in \mathcal{C}$, we exhaustively apply Reduction Rule 1 and 2, where the lowest applicable reduction rule is applied first. Note that each of these reduction rules can be executed in polynomial time, and they can only be applied polynomially many times. If none of the reduction rules are applicable, then we argue below that the number of vertices in $C \setminus D$ can be bounded by $\mathcal{O}(k^3)$. Note that Marking Scheme 1 marks at most $\mathcal{O}(k^3)$ vertices in C. Note that while executing Marking Scheme 2, Reduction Rule 1 is not applicable. Thus the number of vertices marked by Marking Scheme 2 are bounded by $\mathcal{O}(k^3)$ as for every $s \in S$, there can be at most $k+1$ matching edges in both the matchings M_s^1 and M_s^2, and corresponding to each edge we mark $\mathcal{O}(k)$ vertices in C. As the unmarked vertices of $C \setminus D$ are deleted by Reduction Rule 2, we must have $|C \setminus D| \in \mathcal{O}(k^3)$. ◀

References

1 Akanksha Agrawal, Sushmita Gupta, Pallavi Jain, and R. Krithika. Quadratic vertex kernel for split vertex deletion. *Theoretical Computer Science*, 833:164–172, 2020.

2 Akanksha Agrawal, Sudeshna Kolay, Daniel Lokshtanov, and Saket Saurabh. A faster FPT algorithm and a smaller kernel for block graph vertex deletion. In *Theoretical Informatics - 12th Latin American Symposium (LATIN)*, pages 1–13, 2016.

3 Akanksha Agrawal, Daniel Lokshtanov, Pranabendu Misra, Saket Saurabh, and Meirav Zehavi. Feedback vertex set inspired kernel for chordal vertex deletion. *ACM Trans. Algorithms*, 15(1), December 2018. doi:10.1145/3284356.

4 Akanksha Agrawal, Pranabendu Misra, Saket Saurabh, and Meirav Zehavi. Interval vertex deletion admits a polynomial kernel. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '19, pages 1711–1730, USA, 2019. Society for Industrial and Applied Mathematics.

5 Jungho Ahn. A polynomial kernel for 3-leaf power deletion. *Master's Thesis, KAIST, South Korea*, 2020.

6 Jungho Ahn, E. Eiben, O-joung Kwon, and Sang il Oum. A polynomial kernel for 3-leaf power deletion. In *MFCS*, 2020.

7 Jungho Ahn, Eun Jung Kim, and Euiwoong Lee. Towards constant-factor approximation for chordal/distance-hereditary vertex deletion. In *31st International Symposium on Algorithms and Computation (ISAAC 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

8 Yixin Cao and Dániel Marx. Interval deletion is fixed-parameter tractable. *ACM Transactions on Algorithms*, 11(3):21:1–21:35, 2015.

9 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer-Verlag, 2015.

10 Marek Cygan and Marcin Pilipczuk. Split vertex deletion meets vertex cover: New fixed-parameter and exact exponential-time algorithms. *Information Processing Letters*, 113(5):179–182, 2013. doi:10.1016/j.ipl.2013.01.001.

11 Michael Dom, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Error compensation in leaf power problems. *Algorithmica*, 44:363–381, April 2006. doi:10.1007/s00453-005-1180-z.

12 Jack Edmonds. Paths, trees and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

13 Eduard Eiben, Robert Ganian, and O-joung Kwon. A single-exponential fixed-parameter algorithm for distance-hereditary vertex deletion. *Journal of Computer and System Sciences*, 97:121–146, 2018.

14 Martin Charles Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Elsevier, 2004.

15 Edward Howorka. A characterization of ptolemaic graphs. *Journal of Graph Theory*, 5(3):323–331, 1981.

16 Bart M. P. Jansen and Marcin Pilipczuk. Approximation and kernelization for chordal vertex deletion. *SIAM Journal on Discrete Mathematics*, 32(3):2258–2301, 2018.

17 Eun Jung Kim and O-joung Kwon. A polynomial kernel for distance-hereditary vertex deletion. In *Algorithms and Data Structures - 15th International Symposium (WADS)*, volume 10389, pages 509–520, 2017.

18 Jon M. Kleinberg and Éva Tardos. *Algorithm Design*. Addison-Wesley, 2006.

19 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.

20 Dániel Marx. Chordal deletion is fixed-parameter tractable. *Algorithmica*, 57(4):747–768, 2010.

21 Ryuhei Uehara and Yushi Uno. Laminar structure of ptolemaic graphs with applications. *Discrete Appl. Math.*, 157(7):1533–1543, April 2009. doi:10.1016/j.dam.2008.09.006.

22 Mihalis Yannakakis. Node- and edge-deletion np-complete problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC)*, pages 253–264, 1978.

Refuting FPT Algorithms for Some Parameterized Problems Under Gap-ETH

Akanksha Agrawal
Indian Institute of Technology Madras, Chennai, India

Ravi Kiran Allumalla
Indian Institute of Technology Madras, Chennai, India

Varun Teja Dhanekula
Indian Institute of Technology Madras, Chennai, India

Abstract

In this article we study a well-known problem, called Bipartite Token Jumping and not-so-well known problem(s), which we call, Half (Induced-) Subgraph, and show that under Gap-ETH, these problems do not admit FPT algorithms. The problem Bipartite Token Jumping takes as input a bipartite graph G and two independent sets S, T in G, where $|S| = |T| = k$, and the objective is to test if there is a sequence of exactly k-sized independent sets $\langle I_0, I_1, \cdots, I_\ell \rangle$ in G, such that: i) $I_0 = S$ and $I_\ell = T$, and ii) for every $j \in [\ell]$, I_j is obtained from I_{j-1} by replacing a vertex in I_{j-1} by a vertex in $V(G) \setminus I_{j-1}$. We show that, assuming Gap-ETH, Bipartite Token Jumping does not admit an FPT algorithm. We note that this result resolves one of the (two) open problems posed by Bartier et al. (ISAAC 2020), under Gap-ETH. Most of the known reductions related to Token Jumping exploit the property given by triangles (i.e., C_3s), to obtain the correctness, and our results refutes FPT algorithm for Bipartite Token Jumping, where the input graph cannot have any triangles.

For an integer $k \in \mathbb{N}$, the half graph $S_{k,k}$ is the graph with vertex set $V(S_{k,k}) = A_k \cup B_k$, where $A_k = \{a_1, a_2, \cdots, a_k\}$ and $B_k = \{b_1, b_2, \cdots, b_k\}$, and for $i, j \in [k]$, $\{a_i, b_j\} \in E(T_{k,k})$ if and only if $j \geq i$. We also study the Half (Induced-)Subgraph problem where we are given a graph G and an integer k, and the goal is to check if G contains $S_{k,k}$ as an (induced-)subgraph. Again under Gap-ETH, we show that Half (Induced-)Subgraph does not admit an FPT algorithm, even when the input is a bipartite graph. We believe that the above problem (and its negative) result maybe of independent interest and could be useful obtaining new fixed parameter intractability results.

There are very few reductions known in the literature which refute FPT algorithms for a parameterized problem based on assumptions like Gap-ETH. Thus our technique (and simple reductions) exhibits the potential of such conjectures in obtaining new (and possibly easier) proofs for refuting FPT algorithms for parameterized problems.

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

Keywords and phrases Token Jumping, Bipartite Graphs, Fixed Parameter Intractability, Half Graphs, Gap-Exponential Time Hypothesis

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.2

1 Introduction

In this article we study two graph problems and obtain intractability results under Gap-Exponential Time Hypothesis (Gap-ETH), which was conjectured by Dinur [13] and Marx [33]. Gap-ETH conjectures that for some constant $\epsilon > 0$, even to distinguish whether all clauses or at most $(1 - \epsilon)$ fraction of the clauses in the given CNF-SAT formula can be satisfied by an assignment, requires exponential amount of time. Most of the known (fixed parameter) intractability results in the field of parameterized complexity are based on conjectures like FPT $\neq$ W[t], where t is a positive integer. While conjectures like Gap-ETH are stronger

than the more accepted conjectures like the one stated above, they can be used to explain the hardness of a problem by connecting them to SAT, particularly in the absence of insights into the (in)tractability/hardness of a problem. The main message of this article is that, we can (in certain situtation) harness the "approximation gap" provided by the Gap-ETH in obtaining new and simple intractability results. We would like to remark that Gap-ETH has been also previously used by Bhattacharyya et al. [4] for obtaining fixed parameter intractability result for Even Set (among other problems), whose complexity status, prior to their result, was an infamous open problem in the field (see for example, the book of Downey and Fellow [15]).

Reconfigurations and Token Jumping. The first problem that we study is a reconfiguration problem, where in a typical such problem, we have an underlying problem, an instance of the underlying problem, solutions for the instances, and a transformation function from one solution to the other, and the objective is to use check existence of a sequence of transformation from a given solution to another (see, for example, [35] for an introduction to reconfigurations). We study the reconfiguration of independent sets on a graph, where the initial independent set S and final independent set T are of the same size k. The transformations we allow is called token jumping, which was first introduced by Kamiński et al. [22]. We can view a transformation of an independent set to another, in our setting, as jumping a token from one vertex to some other vertex, while maintaining the independence property, where all the tokens are initially placed in the former independent set S. The objective of Token Jumping for a given graph G and independent sets S and T of size k is to test existence of a sequence of transformations from S to T via jumping tokens (one at-a-time). For two sets X and Y, $X \Delta Y$ denotes the set $(X \setminus Y) \cup (Y \setminus X)$. Next we formally define the problem Bipartite Token Jumping, which is Token Jumping when the input is a bipartite graph.

Bipartite Token Jumping	**Parameter:** k
Input: A bipartite graph G and two independent sets S and T in G, where $\|S\| = \|T\| = k$.	
Question: Determine if there a sequence of exactly k-sized independent sets $\langle I_0, I_1, \cdots, I_\ell \rangle$, in G, such that the following holds:	
1. $I_0 = S$ and $I_\ell = T$, and	
2. for each $j \in [\ell]$, $\|I_{j-i} \Delta I_j\| = 2$.	

Some Known Results on Token Jumping. We note that Token Jumping problem is known to be polynomial time solvable, when the input is restricted to special graph classes like trees [11], interval graphs [5], bipartite permutation and bipartite distance-hereditary graphs [17], line graphs [19] and even-hole-free graphs [22].

Although Independent Set on bipartite graphs is one of the classic polynomial time solvable problem (see, for instance, [24] and [18]), Lokshtanov and Mouawad [31] showed that the problem Bipartite Token Jumping is NP-hard. Lokshtanov et al. [32] showed that Token Jumping is W[1]-hard, when parameterized by the sizes of the given independent sets. They also designed an FPT algorithm for the problem when the input graph has bounded degeneracy. We remark that prior to the above result, Ito et al. [21] had obtained an FPT algorithm for the problem when parameterized by k and the maximum degree of the input graph. Token Jumping is also known to be FPT on strongly $K_{\ell,\ell}$-free graphs [6, 20], where a graph is said to be strongly $K_{\ell,\ell}$-free if it does not contain the complete bipartite graph $K_{\ell,\ell}$ as a subgraph. Bartier et al. [2, 3] designed an FPT algorithm for Token Jumping

on C_4-free bipartite graphs when parameterized by the sizes of the independent sets. They also showed that the problem has no FPT algorithm when the input is restricted to C_4-free graphs. The status of Token Jumping on bipartite graphs, i.e., the problem Bipartite Token Jumping, was left as an open problem by Bartier et al. [2].

Our Result on Token Jumping. We resolve one of the open problems of Bartier et al. [2], under Gap-ETH, by proving the following theorem.

▶ **Theorem 1.** *Assuming Gap-ETH, Bipartite Token Jumping does not admit an FPT algorithm, when parameterized by k.*

We obtain the above result by via a simple reduction from the Maximum Balanced Biclique problem, and then use its FPT-inapproximability result by Chalermsook et al. [9] under Gap-ETH, to show that Bipartite Token Jumping cannot have an FPT algorithm. Roughly speaking, by exploiting the Pigeonhole principle in our proof, if the input bipartite graph has a biclique $K_{k,k}$ as an (induced-)subgraph with bipartition A and B, then we are able to argue that there will be a point in the transformation from one of the appropriately constructed independent set to the another, which must contain at least half of vertices from A and B each. Using the above we are able to exploit the known FPT inapproximability of Maximum Balanced Biclique to obtain our result.

(Induced-)Subgraphs and Half Graphs. The Subgraph Isomorphism and Induced-Subgraph Isomorphism are among the fundamental problems in algorithmic graph theory. (Induced-)Subgraph Isomorphism takes as input two graphs G and H, and the objective is to test if (an isomorphic copy of) H is contained as an (induced-)subgraph in G. For a family of graphs Π, a natural generalization of (Induced-)Subgraph Isomorphism is the problem Π (Induced-)Subgraph, where we are given a graph G and an integer k, and the objective is to test if G contains an (induced-)subgraph isomorphic to a graph on k vertices in the family Π. The problems Subgraph Isomorphism and Induced-Subgraph Isomorphism are among the very well-studied problems in Computer Science, which are NP-hard, as they (both) encapsulate problems like Clique (see, [10] and [23]). The fixed parameter intractability of Clique, parameterized by the solution size [14], also implies that both of the above problems do not admit any FPT algorithm when parameterized by $|V(H)|$.

We study Π (Induced-)Subgraph, when Π is a family of structured graphs, called *half* graphs (see, [16]) and the corresponding problem(s) Half (Induced-)Subgraph.[1] For an integer $k \in \mathbb{N}$, the half graph $S_{k,k}$ is the graph with vertex set $V(S_{k,k}) = A_k \cup B_k$, where $A_k = \{a_1, a_2, \cdots, a_k\}$ and $B_k = \{b_1, b_2, \cdots, b_k\}$, and for $i, j \in [k]$, $\{a_i, b_j\} \in E(S_{k,k})$ if and only if $j \geq i$ (see Figure 1, for an illustration).

Our result will particularly focus on the instances where the input graph is bipartite, and thus we formally define the problems Bipartite Half Subgraph and Bipartite Half Induced-Subgraph below.

Bipartite Half (Induced-)Subgraph	**Parameter:** k
Input: A bipartite graph G and an integer k.	
Question: Does G contain a graph isomorphic to $S_{k,k}$ as an (induced-)subgraph?	

[1] We note that the term "skew" was used in the context of Skew Multicut and Directed Feedback Arc Set for a graph class closely related to half graphs (see [26], for more details). Another closely related graph class is the family of bipartite chain graphs (or bisplits), please see [7].

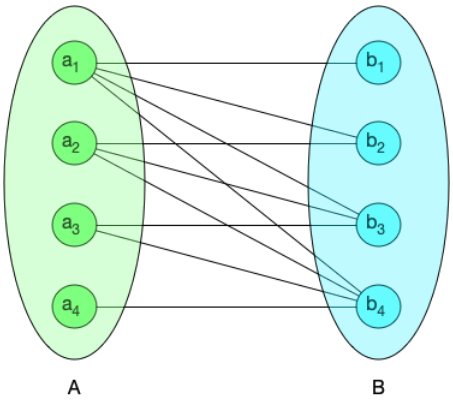


Figure 1 The half graph $S_{4,4}$.

Some Known Results on Π (Induced-)Subgraph**.** The problems Π (Induced-)Subgraph have been studied for several structured families of graphs, most notable being Clique and Independent Set. Although, when Π is among the many well-known and highly structured families like cliques, independent sets, and bicliques, the problem seem to be "inherently-enumerative", and thus, fixed parameter intractable [9], we do know several instances where the problem is FPT. Khot and Raman [25] obtained a complete fixed parameter tractability dichotomy for Π Subgraph, for hereditary families, by showing that if Π includes all graphs with no edges but not all complete graphs, or vice versa, then the problem is fixed parameter intractable, and otherwise, it admits an FPT algorithm. Also, Kratsch et al. [27] studied the kernelization complexity for hereditary families of graphs for such problems. Lin and Chen [30] obtained an FPT algorithm for Π Induced-Subgraph, when Π is the family of all graphs with exactly k edges.[2]

One of the notorious parameterized problems, whose status of fixed parameter tractability was open until 2015 is the Biclique problem. Despite it's deceptive similarity with Clique, the problem was not known to be fixed parameter intractable, until such a result was obtained by Lin [28, 29]. Later, Chalermsook et al. [8, 9] showed that, for any function $f \in o(k)$, Biclique does not admit a $k/f(k)$-approximation algorithm, running in FPT time, assuming Gap-ETH.

Our Contribution on Π (Induced-)Subgraph**.** We study the Bipartite Half (Induced-) Subgraph problem(s), and obtain the following result.

▶ **Theorem 2.** *Assuming Gap-ETH, Bipartite Half (Induced-)Subgraph does not admit an FPT algorithm, when parameterized by k.*

We obtain the above result by a (Turing) reduction starting from the problem Maximum Balanced Biclique. For the given instance of Maximum Balanced Biclique, we begin by "color-coding" a copy of $K_{k,k}$ in the input graph (if it exist). We note that color-coding was introduced by Alon et al. [1], for designing FPT algorithms for parameterized by the solution size, for detecting cycles and paths of a given length. After this we create two copies of each of the color classes obtained using the color-coding to "complete" an $S_{2k,2k}$ using the

[2] Here, k used in the definition of Π is the same as the number k which is provided as input in the Π Induced-Subgraph instance.

presence of the biclique. We note that directly using Multicolored Biclique leads to issues in the one of direction of the proof, relating to the stricter condition of picking vertices from each of the color classes. We note that approximation in the solution that we are able to output for the Maximum Balanced Biclique instance arises due to the possibility of picking multiple copies of the same vertex.

We would like to note that, we came across the problem Bipartite Half (Induced-) Subgraph, while studying Bipartite Token Jumping. Although, later we obtained a simple proof for our negative result regarding Bipartite Token Jumping, eliminating the need to go via the route of Bipartite Half (Induced-)Subgraph. Nevertheless we believe due to the "sequence-like" structure it offers, it may turn out to be a useful starting point for obtaining new intractability results, similar to its cousins like Clique and Biclique.

2 Preliminaries

Sets and Functions. We denote the set of natural numbers (including 0) by $\mathbb{N}$. For $n \in \mathbb{N}$, we use $[n]$ and $[n]_0$ to denote the sets $\{1, 2, \cdots, n\}$ and $\{0, 1, 2, \cdots, n\}$, respectively. For a set X, we denote its power set by $2^X = \{X' \mid X' \subseteq X\}$. For a set Y, by $Y \times Y$, we denote the set $\{(y, y') \mid y, y' \in Y\}$. For a function $f : X \to Y$ and an element $y \in Y$, $f^{-1}(y)$ denotes the set $\{x \in X \mid f(x) = y\}$. For a non-empty set X, a family $\mathcal{F} \subseteq 2^X$ is a *partition* of X, if i) for each $Y \in \mathcal{F}$, $Y \neq \emptyset$, ii) for distinct $Y, Z \in \mathcal{F}$, we have $Y \cap Z = \emptyset$, and iii) $\cup_{Y \in \mathcal{F}} Y = X$.

Graph Theory. We refer to the book of [12] for standard graph terminology. Given a graph G, we denote its vertex set and edge set by $V(G)$ and $E(G)$, respectively. Whenever the context is clear, we use n and m to denote the number of vertices and the number of edges in the input graph, respectively. Consider a graph G. For $X \subseteq V(G)$, $G[X]$ denotes the subgraph of G induced by X, i.e. $G[X]$ is the graph with vertex set X and edge set $\{\{x, y\} \in E(G) \mid x, y \in X\}$.

For a graph G, a set $S \subseteq V(G)$ is an *independent set* in G if for every $u, v \in S$, we have $\{u, v\} \notin E(G)$. A graph G is *bipartite* if its vertex set can be partitioned into two independent sets X and Y. In the above, X, Y is called a *bipartition* of G.

For an integer $k \in \mathbb{N}$, $K_{k,k}$ denotes the bipartite graph with a bipartition A and B, each with k vertices, so that for each $a \in A$ and $b \in B$, we have $\{a, b\} \in E(K_{k,k})$. In the above, $K_{k,k}$ is called a *biclique*. For a bipartite graph G, by $\mathsf{opt\text{-}biclq}(G)$ we denote the largest integer k^*, such that G has a K_{k^*,k^*}-biclique as an induced subgraph.

Satisfiability and Gap-ETH. For $q \in \mathbb{N}$, the q-SAT problem takes as input a q-CNF formula ϕ with m clauses on a variable set X of size n, and the objective is to test if there is an assignment $\mathsf{asg} : X \to \{0, 1\}$ that satisfies ϕ.

Next we define the maximization version of q-SAT, called Max q-SAT. For $q \in \mathbb{N}$, the problem Max q-SAT takes as input a q-CNF formula ϕ with m clauses on a variable set X of size n, and the objective is to compute, $\mathsf{opt\text{-}SAT}(\phi)$, which is the maximum number of clauses in ϕ that can be simultaneously satisfied by an assignment $\mathsf{asg} : X \to \{0, 1\}$.

We next state the statement of Gap-ETH.

▶ **Conjecture 3** (Gap-Exponential Time Hypothesis (Gap-ETH), [13, 33]). *For some constant $\delta, \epsilon > 0$, no algorithm can, given a 3-CNF formula ϕ on n variables and $m = \mathcal{O}(n)$ clauses, distinguish between the following cases correctly with probability at least $2/3$ in time $\mathcal{O}(2^{\delta n})$: i) $\mathsf{opt\text{-}SAT}(\phi) = m$ and ii) $\mathsf{opt\text{-}SAT}(\phi) < (1 - \epsilon) \cdot m$.*

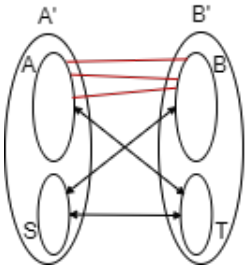


Figure 2 Illustration of construction of G'. Double arrows show all possible edges between the sets, and set lines between A and B shows complementation of edges between them from G.

Some Results on Maximum Balanced Biclique. In the Maximum Balanced Biclique problem we are given a bipartite graph G and an integer k, and the objective is to test if $\mathsf{opt\text{-}biclq}(G) \geq k$.[3]

In the following we state an inapproximability result regarding Maximum Balanced Biclique, which follows as a corollary from Definition 2.1 and 3.2, Proposition 3.3 and Corollary 5.16 of [9].

▶ **Proposition 4.** *Assuming Gap-ETH, there is no algorithm, which for all bipartite graphs G on n vertices and an integer k, runs in time $f(k) \cdot n^{\mathcal{O}(1)}$, and does one of the following i) outputs 1 if $\mathsf{opt\text{-}biclq}(G) \geq k$, or ii) outputs 0 if $\mathsf{opt\text{-}biclq}(G) < k/2$; where f is some computable function. That is, Maximum Balanced Biclique does not admit a factor 2-approximation algorithm running in FPT time.*

We remark that the above proposition is a very special instantiation, which is enough for our purpose, of the more general result of Corollary 5.16 of [9].

3 Refuting FPT Algorithms for Bipartite Token Jumping

The objective of this section is to prove Theorem 1. We obtain the above by exhibiting an appropriate reduction from Maximum Balanced Biclique. We begin by explaining the intuition behind our reduction. Consider an instance (G, k) of Maximum Balanced Biclique, where G is a bipartite graph with vertex bipartition A and B. We will create an instance (G', S, T) of Bipartite Token Jumping as follows. Intuitively speaking, G' will be obtained from G by adding two new sets of vertices S and T, each of size k. The construction will ensure that in any transformation from S to T, there must exist a consecutive pair of independent sets, which places $k/2$ tokens in A and B, each, which will induce a complete bipartite graph in G (via complementing edges). We will now formally describe our construction.

Construction of (G', S, T). Initialize $V(G') = V(G) = A \cup B$ and $E(G') = \{\{a, b\} \mid a \in A, b \in B, \text{ and } \{a, b\} \notin E(G)\}$ (see Figure 2). We add two sets of $(k+1$ new vertices), $S = \{s_1, s_2, \cdots, s_k, s_{k+1}\}$ and $T = \{t_1, t_2, \cdots, t_k, t_{k+1}\}$ to $V(G')$. We add all the edges between $S \cup A$ and T, i.e, we add all the edges in $\{\{u, t\} \mid u \in S \cup A \text{ and } t \in T\}$ to $E(G')$. Also,

[3] The definition of Maximum Balanced Biclique given in [9], is a purely optimization version, where k is not part of the input. We state the problem this way, as this helps us get around unnecessary technicalities, without compromising on the technical merit of the result.

we add all the edges between vertices in S and B, i.e., the edges in $\{\{s,b\} \mid s \in S \text{ and } b \in B\}$ to $E(G')$. Clearly, by the construction, G' is a bipartite graph, with bipartition $A' = S \cup A$ and $B' = T \cup B$. The above concludes the construction of the Bipartite Token Jumping instance (G', S, T).

In the following lemmata we will establish some useful properties that will help us prove the theorem.

▶ **Lemma 5.** *If (G,k) is a yes-instance of Maximum Balanced Biclique, then (G', S, T) is a yes-instance of Bipartite Token Jumping.*

Proof. Suppose that $\mathsf{opt\text{-}biclq}(G) \geq k$, and consider $X = \{x_1, x_2, \cdots, x_k\} \subseteq A$ and $Y = \{y_1, y_2, \cdots, y_k\} \subseteq B$, such that $G[X \cup Y]$ is isomorphic to $K_{k,k}$. We will construct a sequence $\langle S = I_0, I_1, \cdots, I_k, I'_1, I'_2, \cdots, I'_k, \widehat{I}_1, \widehat{I}_2, \cdots, \widehat{I}_{k+1} = T\rangle$, of independent sets of size $k+1$ in G', to conclude that (G', S, T) is a yes-instance of Bipartite Token Jumping.

For $j \in [k]$, let $I_j = (I_{j-1} \setminus \{s_j\}) \cup \{x_j\}$. As $X \subseteq A$ and A is an independent set in G', it follows that, for each $j \in [k]$, I_j is an independent set in G'. Recall that S is an independent set in G' of size exactly $k+1$, $S \cap A = \emptyset$, and $X \subseteq A$. Thus we can obtain that, for each $j \in [k]$, I_j is an independent set in G' of size exactly $k+1$ in G'.

Let $I'_1 = (I_k \setminus \{s_{k+1}\}) \cup \{y_1\}$. For $j \in [k] \setminus \{1\}$, $I'_j = (I'_{j-1} \setminus \{x_{j-1}\}) \cup \{y_j\}$. Note that for each $x \in X$ and $y \in Y$, we have $\{x,y\} \in E(G)$, and thus $\{x,y\} \notin E(G')$. The above together with the fact that $X \cap Y = \emptyset$, implies that for each $j \in [k]$, I'_j is an independent set in G' of size exactly $k+1$.

Finally, let $\widehat{I}_1 = (I'_k \setminus \{x_k\}) \cup \{t_1\}$, and for $j \in [k+1] \setminus \{1\}$, let $\widehat{I}_j = (\widehat{I}_{j-1} \setminus \{y_{j-1}\}) \cup \{t_j\}$. As $B \cup T$ is an independent set in G' and I'_k is an independent set of size exactly $k+1$ in G', it follows that for each $j \in [k+1]$, $\widehat{I}_j$ is an independent set of exactly size $k+1$ in G'. Also, by the construction of $\widehat{I}_{k+1}$ it follows that $\widehat{I}_{k+1} = T$.

From the construction of $\langle S = I_0, I_1, \cdots, I_k, I'_1, I'_2, \cdots, I'_k, \widehat{I}_1, \widehat{I}_2, \cdots, \widehat{I}_{k+1} = T\rangle$ it follows that it is a solution for the Bipartite Token Jumping instance (G, S, T). This concludes the proof. ◀

In the next lemma we exploit a solution for the Bipartite Token Jumping instance (if it exists), to obtain an approximation for the Maximum Balanced Biclique instance.

▶ **Lemma 6.** *If (G', S, T) is a yes-instance of Bipartite Token Jumping, then* $\mathsf{opt\text{-}biclq}(G) \geq k/2$.

Proof. Suppose that (G', S, T) is a yes-instance of Bipartite Token Jumping, and let $\langle S = I_0, I_1, \cdots, I_\ell = T\rangle$ be a solution for it. Recall that for each $s \in S$ and $u \in T \cup B$, we have $\{s,u\} \in E(G')$, $|S| = |T| = k+1$, and $S \cap T = \emptyset$. Thus, there must exist $q \in [\ell]$ such that $|I_q \cap A| = k$. Let q_{lrg} be the largest integer in $[\ell]$, such that $|I_{q_{\mathsf{lrg}}} \cap A| = k$. Let r_{sml} be the smallest integer in $r \in [\ell]$, such that $r_{\mathsf{sml}} > q_{\mathsf{lrg}}$ and $|I_r \cap (T \cup B)| \geq k$. Note that r_{sml} exists as $T = I_\ell$. As $|I_{q_{\mathsf{lrg}}} \cap A| = k$, $|I_{r_{\mathsf{sml}}} \cap (T \cup B)| = k$, and $A \cap (T \cup B) = \emptyset$, there must exist $j \in [\ell]$, such that $q_{\mathsf{lrg}} < j < r_{\mathsf{sml}}$ and $|I_j \cap (T \cup B)| = \lceil k/2 \rceil$. As $k \geq 2$, $I_j \cap (T \cup B) \neq \emptyset$, and thus there is some $v \in I_j \cap (T \cup B)$. Also, $|I_j| = k+1$, and thus, I_j must contain at least $k + 1 - \lceil k/2 \rceil \geq k/2$ vertices from $S \cup A$. Recall that by the construction of G', $\{s,v\} \in E(G')$, for each $s \in S$. From the above we can obtain that $I_j \cap S = \emptyset$. Recall that each vertex in T is adjacent to every vertex in $S \cup A$. From the above discussions we can obtain that $X = I_j \cap A$ has at least $k/2$ vertices and $Y = I_j \cap B$ has at least $\lceil k/2 \rceil$ vertices. By the construction of G', we can obtain that $G[X \cup Y]$ is a biclique in G, and thus we can obtain that $\mathsf{opt\text{-}biclq}(G) \geq k/2$. ◀

We are now ready to prove Theorem 1.

Proof of Theorem 1. Towards a contradiction, suppose that there is an algorithm $\mathcal{A}$, for Bipartite Token Jumping running in time $f(k)n^{\mathcal{O}(1)}$. Using $\mathcal{A}$, we design an algorithm $\mathcal{B}$, running in time $f(k)n^{\mathcal{O}(1)}$, which given a bipartite graph G and an integer k, returns 1 if $\mathsf{opt\text{-}biclq}(G) \geq k$, and returns 0 if $\mathsf{opt\text{-}biclq}(G) < k/2$, thus contradicting Proposition 4.

Our algorithm $\mathcal{B}$ is as follows. Given an instance (G, k) of Maximum Balanced Biclique, $\mathcal{B}$ constructs (in polynomial time) an instance (G', S, T) of Bipartite Token Jumping, as discussed previously. The algorithm $\mathcal{B}$ calls $\mathcal{A}$ on the input (G', S, T), and returns 1 if and only if $\mathcal{A}$ returns 1 on the Bipartite Token Jumping instance.

We now argue the correctness of our algorithm. We first argue that whenever $\mathsf{opt\text{-}biclq}(G) \geq k$, then $\mathcal{B}$ returns 1. From Lemma 5, whenever $\mathsf{opt\text{-}biclq}(G) \geq k$, then (G', S, T) is a yes-instance of Bipartite Token Jumping. Thus $\mathcal{A}$, and hence $\mathcal{B}$, must return 1. Notice that the contrapositive of Lemma 6 gives us that whenever $\mathsf{opt\text{-}biclq}(G) < k/2$, then (G', S, T) is a no-instance of Bipartite Token Jumping. Thus, $\mathcal{A}$ and hence, $\mathcal{B}$ must return 0 for the above case. This concludes the proof. ◀

4 Refuting FPT Algorithms for Bipartite Half (Induced-)Subgraph

The objective of this section is to prove Theorem 2. We begin with the following simple observation that will be useful in "encoding" an instance Maximum Balanced Biclique as instances of Bipartite Half Induced-Subgraph (resp. Bipartite Half Subgraph).

▶ **Observation 7.** *For any $k \in \mathbb{N}$, the half graph $T_{2k,2k}$ contains $K_{k,k}$ as an induced subgraph.*

Proof. Recall that, for any $k \in \mathbb{N}$, $T_{2k,2k}$ is the graph with vertex set $V(T_{2k,2k}) = A_{2k} \cup B_{2k}$, where $A_{2k} = \{a_1, a_2, \cdots, a_{2k}\}$ and $B_{2k} = \{b_1, b_2, \cdots, b_{2k}\}$, and for $i, j \in [2k]$, $\{a_i, b_j\} \in E(T_{2k,2k})$ if and only if $j \geq i$. From the above we can deduce that, for any $i \in [k]$ and $j \in [2k] \setminus [k]$, we have $\{a_i, b_j\} \in E(T_{k,k})$. The above implies that $T_{2k,2k}[\{a_i \mid i \in [k]\} \cup \{b_j \mid j \in [2k] \setminus [k]\}]$ is isomorphic to $K_{k,k}$. ◀

We will use the above observation to encode (approximately) the instance of Maximum Balanced Biclique into an instance of Bipartite Half (Induced-)Subgraph. We remark that the same construction will work for both Bipartite Half Subgraph and Bipartite Half Induced-Subgraph. Before moving further we introduce a definition and an algorithmic result about it, that will be used in our reduction.

▶ **Definition 8.** For $n, \ell \in \mathbb{N}$, a family $\mathcal{F}$ of functions from $[n]$ to $[\ell]$ is an (n, ℓ)-*perfect hash family* if for every $S \subseteq [n]$ of size ℓ, there exists $f \in \mathcal{F}$, such that for each $i \in [\ell]$, $|S \cap \{j \in [n] \mid f(j) = i\}| = 1$.

▶ **Proposition 9** ([34]). *For any $n, \ell \in \mathbb{N}$, we can construct an (n, ℓ)-prefect hash family of size $e^{\ell} \cdot \ell^{\mathcal{O}(\log \ell)} \cdot \log n$ in time $e^{\ell} \cdot \ell^{\mathcal{O}(\log \ell)} \cdot n \log n$.*

We will next intuitively explain our (Turing) reduction below. Consider an instance (G, k) of Maximum Balanced Biclique, where G is a bipartite graph with bipartition A and B. We will begin by "color-coding" a copy of $K_{k,k}$ in G (if it exists). We remark that the technique of color-coding was introduced by Alon et al. [1], for designing FPT algorithms parameterized by the solution size, for detecting cycles and paths of a given length. Also, the above was derandomized (in the same paper), by using the construction of perfect-hash families based on the result of Schmidt and Siegel [36], and an improved

algorithm for computing such families was obtained by Naor et al. [34] (see Proposition 9). The idea of color coding is to distinctly color the vertices in the copy of $K_{k,k}$ present (as a subgraph/induced subgraph) in G. Using the above, we obtain a partition of A into k sets $A_1, A_2, \cdots, A_k$, and also a partition of B into k sets, $B_1, B_2, \cdots, B_k$, so that the vertices of a fixed $K_{k,k}$ in G picks exactly one vertex from each A_is and B_is. After this, using one more copy of the sets A_is and B_is, to be denoted by A'_is and B'_is, respectively, we create an instance of Bipartite Half (Induced-)Subgraph, by providing extra vertices/edges using the copies. We remark that the "loss" in our reduction comes from the situation where a solution for the Bipartite Half (Induced-)Subgraph instance may pick two copies of the same vertex.

We will now move to the formal description of the reduction. Consider an instance (G, k) of Maximum Balanced Biclique, where G is a bipartite graph with bipartition $A = \{a_1, a_2, \cdots, a_{n_1}\}$ and $B = \{b_1, b_2, \cdots, b_{n_2}\}$, and let $n = n_1 + n_2$. We let $A' = \{a'_i \mid i \in [n_1]\}$ and $B' = \{b'_i \mid i \in [n_2]\}$, copies of A and B, respectively. Using Proposition 9, we begin by computing (n_1, k)- and (n_2, k)-prefect hash families, $\mathcal{F}_1$ and $\mathcal{F}_2$, respectively. For $f \in \mathcal{F}_1$ and $i \in [k]$, we let $A[f, i] = \{a_j \in A \mid f(j) = i\}$ and $A'[f, i] = \{a'_j \in A' \mid f(j) = i\}$. Similarly, for $f \in \mathcal{F}_2$ and $i \in [k]$, we let $B[f, i] = \{b_j \in B \mid f(j) = i\}$ and $B'[f, i] = \{b'_j \in B' \mid f(j) = i\}$. We will use the convention that, for a vertex $v \in A \cup B$, its corresponding copy in $A' \cup B'$ will be denoted by v'.

For each pair of functions $f_1 \in \mathcal{F}_1$ and $f_2 \in \mathcal{F}_2$, we create a Bipartite Half (Induced-) Subgraph instance $I_{f_1,f_2} = (G_{f_1,f_2}, 2k)$, where G_{f_1,f_2} is a bipartite graph as follows. We let $V(G_{f_1,f_2}) = A \cup B \cup A' \cup B'$. We will now describe the edge set of G_{f_1,f_2}. First we add all the edges in G, between A and B, to G_{f_1,f_2}, i.e., for each $a \in A$ and $b \in B$, such that $\{a, b\} \in E(G)$, we add $\{a, b\}$ to $E(G_{f_1,f_2})$. (Intuitively speaking, the objective of the above edges will be to directly preserve, in G_{f_1,f_2}, the $K_{k,k}$s of G, if they exist.) Roughly speaking, next we will add edges, that will be useful in completing the missing vertices/edges in a half graph (using the copies A' and B'), apart from those that are already present in the biclique. For each $i \in [k]$ and $j \in [k]$, where $j \geq i$, and for each $a \in A[f_1, i]$ and $b \in B[f_2, j]$, we add $\{a, b'\}$ to $E(G_{f_1,f_2})$ if and only if $\{a, b\} \in E(G)$. Similarly, for each $i \in [k]$ and $j \in [k]$, where $j \geq i$, and for each $a \in A[f_1, i]$ and $b \in B[f_2, j]$, we add $\{a', b\}$ to $E(G_{f_1,f_2})$ if and only if $\{a, b\} \in E(G)$. This completes the description of G_{f_1,f_2}, and thus the Bipartite Half (Induced-)Subgraph instance $I_{f_1,f_2} = (G_{f_1,f_2}, 2k)$. We let $\mathcal{I} = \{I_{f_1,f_2} \mid f_1 \in \mathcal{F}_1 \text{ and } f_2 \in \mathcal{F}_2\}$.

In the following two lemmata we will establish some useful relationships between the Maximum Balanced Biclique instance (G, k) and the instances of Bipartite Half (Induced-)Subgraph that we have created.

▶ **Lemma 10.** *If (G, k) is a yes-instance of Maximum Balanced Biclique, then there are $f_1 \in \mathcal{F}_1$ and $f_2 \in \mathcal{F}_2$, such that I_{f_1,f_2} is a yes-instance of Bipartite Half (Induced-)Subgraph.*

Proof. Suppose that $\mathsf{opt\text{-}biclq}(G) \geq k$, and consider $X = \{x_1, x_2, \cdots, x_k\} \subseteq A$ and $Y = \{y_1, y_2, \cdots, y_k\} \subseteq B$, such that $G[X \cup Y]$ is isomorphic to $K_{k,k}$. As $\mathcal{F}_1$ is an (n_1, k)-perfect hash family, there must exist $f_1^* \in \mathcal{F}_1$, such that for each $i \in [k]$, $|A[f_1^*, i] \cap X| = |A'[f_1^*, i] \cap X| = 1$. Similarly, there exists $f_2^* \in \mathcal{F}_2$, such that for each $i \in [k]$, $|B[f_2^*, i] \cap Y| = |B'[f_2^*, i] \cap Y| = 1$. Without loss of generality, we will assume that for $i \in [k]$, we have $A[f_1^*, i] \cap X = \{x_i\}$ and $B[f_2^*, i] \cap X = \{y_i\}$. (Notice that the above can be achieved by renaming of the vertices.) Let $X^* = \{x_i \mid i \in [k]\} \cup \{x'_i \mid i \in [k]\}$ and $Y^* = \{y_i \mid i \in [k]\} \cup \{y'_i \mid i \in [k]\}$, $G_{f_1^*,f_2^*}[X^* \cup Y^*]$. The construction of $G_{f_1^*,f_2^*}$ and the assumption that $G[X \cup Y]$ is isomorphic to $K_{k,k}$, implies that $G_{f_1^*,f_2^*}[X^* \cup Y^*]$ is isomorphic to $T_{2k,2k}$. This concludes the proof. ◀

▶ **Lemma 11.** *If there are functions $f_1 \in \mathcal{F}_1$ and $f_2 \in \mathcal{F}_2$, such that $(G_{f_1,f_2}, 2k)$ is a yes-instance of* Bipartite Half (Induced-)Subgraph, *then* opt-biclq$(G) \geq k/2$.

Proof. Consider $f_1 \in \mathcal{F}_1$ and $f_2 \in \mathcal{F}_2$, such that $(G_{f_1,f_2}, 2k)$ is a yes-instance of Bipartite Half (Induced-)Subgraph. Let $X^* \subseteq A \cup A'$ and $Y^* \subseteq B \cup B'$ be each of size $2k$, such that $G_{f_1,f_2}[X^* \cup Y^*]$ contains $T_{2k,2k}$ as an (induced) subgraph.[4] From Observation 7, we can obtain that there is $\widehat{X} \subseteq X^* \subseteq A \cup A'$ and $\widehat{Y} \subseteq Y^* \subseteq B \cup B'$, each of size k, such that $G_{f_1,f_2}[\widehat{X} \cup \widehat{Y}]$ is isomorphic to $K_{k,k}$. We construct two sets $X \subseteq A$ and $Y \subseteq B$ as follows, each initialised to $\emptyset$. For each $a \in A$, we add a to X, if $\{a, a'\} \cap \widehat{X} \neq \emptyset$. Similarly, for each $b \in B$, we add b to Y, if $\{b, b'\} \cap \widehat{Y} \neq \emptyset$. Note that the sizes of X and Y, each, must be at least $k/2$. Using the above together with the construction of G_{f_1,f_2} can be used to obtain that $G[X \cup Y]$ is isomorphic to $K_{\lceil k/2 \rceil, \lceil k/2 \rceil}$. Thus we conclude that opt-biclq$(G) \geq k/2$. ◀

Now we are ready to prove Theorem 2.

Proof of Theorem 2. Towards a contradiction, suppose that there is an algorithm $\mathcal{A}$, for Bipartite Half (Induced-)Subgraph running in time $f(k)n^{\mathcal{O}(1)}$, where n is the number of vertices in the input graph. Using $\mathcal{A}$, we design an algorithm $\mathcal{B}$, running in time $g(k) \cdot n^{\mathcal{O}(1)}$ (where g is a computable function), which given a bipartite graph G on n vertices and an integer k, returns 1 if opt-biclq$(G) \geq k$, and returns 0 if opt-biclq$(G) < k/2$, thus contradicting Proposition 4.

Our algorithm $\mathcal{B}$ is as follows. Given an instance (G, k) of Maximum Balanced Biclique, the algorithm $\mathcal{B}$ starts by constructing the family of Bipartite Half (Induced-) Subgraph instance $\mathcal{I}$, as discussed previously, in time bounded by $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(1)}$ time (see Proposition 9). Next, for each $I \in \mathcal{I}$, $\mathcal{B}$ calls $\mathcal{A}$ for the input I of Bipartite Half (Induced-)Subgraph. The algorithm $\mathcal{B}$ return 1 if and only if $\mathcal{A}$ returns 1 for at least one instance $I \in \mathcal{I}$.

The construction of the set $\mathcal{I}$, together with the assumed running time of $\mathcal{A}$ implies that the running time of $\mathcal{B}$ can be bounded by $f(k) \cdot 2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(1)}$. We now argue the correctness of our algorithm $\mathcal{B}$.

We first argue that whenever opt-biclq$(G) \geq k$, then $\mathcal{B}$ returns 1. From Lemma 10, whenever opt-biclq$(G) \geq k$, then there must exist a yes-instance of Bipartite Half (Induced-)Subgraph $I \in \mathcal{I}$. Thus, in the above case, $\mathcal{B}$ always returns 1, as required. Notice that the contrapositive of Lemma 11 gives us that whenever opt-biclq$(G) < k/2$, then there is no $I \in \mathcal{I}$, such that I is a yes-instance of Bipartite Half (Induced-)Subgraph, and thus, $\mathcal{B}$ will necessarily output 0 for this case. This concludes the proof. ◀

5 Conclusion

Assuming Gap-ETH, we showed that the Bipartite Token Jumping and Bipartite Half (Induced-)Subgraph problems do not admit FPT algorithms. Our results are obtained by appropriate reductions from the Maximum Balanced Biclique problem, and then exploiting the known FPT-inapproximability result of [9] under Gap-ETH, to show that our problems cannot have an FPT algorithms. The natural open problems that arise from our results (and also from the previous works) is obtaining fixed parameter intractability results for these problems under standard and more well-believed conjectures like FPT $\neq$ W$[t]$, where $t \in \mathbb{N} \setminus \{0\}$.

[4] We assume $T_{2k,2k}$ is an induced subgraph of $G_{f_1,f_2}[X^* \cup Y^*]$ when we are considering the problem Bipartite Half Induced-Subgraph, whereas, we assume that it is a subgraph when we are considering the problem Bipartite Half Subgraph.

We believe that our (negative) result regarding Bipartite Half (Induced-)Subgraph maybe a useful starting point for obtaining newer results, similar to its counterparts, like Clique and Biclique. Again we note that not many (parameterized) reductions are known, which refute FPT algorithms under conjectures like Gap-ETH, one of the exceptions to the above is the result of [4]. We believe that newer conjectures like Gap-ETH maybe useful in obtaining newer (and perhaps easier) fixed parameter intractability results, and our reduction serves as a (yet another) evidence for it.

References

1 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995.

2 Valentin Bartier, Nicolas Bousquet, Clément Dallard, Kyle Lomer, and Amer E. Mouawad. On girth and the parameterized complexity of token sliding and token jumping. In *31st International Symposium on Algorithms and Computation (ISAAC)*, volume 181, pages 44:1–44:17, 2020.

3 Valentin Bartier, Nicolas Bousquet, Clément Dallard, Kyle Lomer, and Amer E. Mouawad. On girth and the parameterized complexity of token sliding and token jumping. *Algorithmica*, 83(9):2914–2951, 2021.

4 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)*, volume 107, pages 17:1–17:15, 2018.

5 Marthe Bonamy and Nicolas Bousquet. Token sliding on chordal graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 127–139. Springer, 2017.

6 Nicolas Bousquet, Arnaud Mary, and Aline Parreau. Token jumping in minor-closed classes. In *International Symposium on Fundamentals of Computation Theory*, pages 136–149. Springer, 2017.

7 Andreas Brandstädt, Peter L Hammer, Vadim V Lozin, et al. Bisplit graphs. *Discrete Mathematics*, 299(1-3):11–32, 2005.

8 Parinya Chalermsook, Marek Cygan, Guy Kortsarz, Bundit Laekhanukit, Pasin Manurangsi, Danupon Nanongkai, and Luca Trevisan. From gap-eth to fpt-inapproximability: Clique, dominating set, and more. In *58th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 743–754, 2017.

9 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 Journal on Computing*, 49(4):772–810, 2020.

10 Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing (STOC)*, pages 151–158, 1971.

11 Erik D Demaine, Martin L Demaine, Eli Fox-Epstein, Duc A Hoang, Takehiro Ito, Hirotaka Ono, Yota Otachi, Ryuhei Uehara, and Takeshi Yamada. Polynomial-time algorithm for sliding tokens on trees. In *International Symposium on Algorithms and Computation*, pages 389–400. Springer, 2014.

12 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

13 Irit Dinur. Mildly exponential reduction from gap 3sat to polynomial-gap label-cover. *ECCC*, 23:128, 2016.

14 Rodney G Downey and Michael R Fellows. Fixed-parameter intractability. In *1992 Seventh Annual Structure in Complexity Theory Conference*, pages 36–37. IEEE Computer Society, 1992.

15 Rodney G Downey and Michael R Fellows. *Fundamentals of parameterized complexity*, volume 4. Springer, 2013.

16 Paul Erdös and András Hajnal. Chromatic number of finite and infinite graphs and hypergraphs. *Discrete Mathematics*, 53:281–285, 1985.

17 Eli Fox-Epstein, Duc A Hoang, Yota Otachi, and Ryuhei Uehara. Sliding token on bipartite permutation graphs. In *International Symposium on Algorithms and Computation*, pages 237–247. Springer, 2015.

18 John E Hopcroft and Richard M Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, 2(4):225–231, 1973.

19 Takehiro Ito, Erik D Demaine, Nicholas JA Harvey, Christos H Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412(12-14):1054–1065, 2011.

20 Takehiro Ito, Marcin Kamiński, and Hirotaka Ono. Fixed-parameter tractability of token jumping on planar graphs. In *International Symposium on Algorithms and Computation*, pages 208–219. Springer, 2014.

21 Takehiro Ito, Marcin Kaminski, Hirotaka Ono, Akira Suzuki, Ryuhei Uehara, and Katsuhisa Yamanaka. On the parameterized complexity for token jumping on graphs. In *Theory and Applications of Models of Computation - 11th Annual Conference (TAMC)*, volume 8402, pages 341–351, 2014.

22 Marcin Kamiński, Paul Medvedev, and Martin Milanič. Complexity of independent set reconfigurability problems. *Theoretical computer science*, 439:9–15, 2012.

23 Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

24 Dénes Kőnig. Graphok és alkalmazásuk a determinánsok és a halmazok elméletére. *Mathematikai és Természettudományi Ertesito*, 34:104–119, 1916.

25 Subhash Khot and Venkatesh Raman. Parameterized complexity of finding subgraphs with hereditary properties. *Theoretical Computer Science*, 289(2):997–1008, 2002.

26 Stefan Kratsch, Shaohua Li, Dániel Marx, Marcin Pilipczuk, and Magnus Wahlström. Multi-budgeted directed cuts. In *13th International Symposium on Parameterized and Exact Computation (IPEC)*, volume 115, pages 18:1–18:14, 2018.

27 Stefan Kratsch, Marcin Pilipczuk, Ashutosh Rai, and Venkatesh Raman. Kernel lower bounds using co-nondeterminism: Finding induced hereditary subgraphs. In *Algorithm Theory - SWAT*, volume 7357, pages 364–375, 2012.

28 Bingkai Lin. The parameterized complexity of k-biclique. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 605–615, 2015.

29 Bingkai Lin. The parameterized complexity of the k-biclique problem. *Journal of the ACM (JACM)*, 65(5):1–23, 2018.

30 Bingkai Lin and Yijia Chen. The parameterized complexity of k-edge induced subgraphs. In *International Colloquium on Automata, Languages, and Programming*, pages 641–652. Springer, 2012.

31 Daniel Lokshtanov and Amer E Mouawad. The complexity of independent set reconfiguration on bipartite graphs. *ACM Transactions on Algorithms (TALG)*, 15(1):1–19, 2018.

32 Daniel Lokshtanov, Amer E Mouawad, Fahad Panolan, MS Ramanujan, and Saket Saurabh. Reconfiguration on sparse graphs. *Journal of Computer and System Sciences*, 95:122–131, 2018.

33 Dániel Marx. Parameterized complexity and approximation algorithms. *Computer Journal*, 51(1):60–78, 2008.

34 Moni Naor, Leonard J. Schulman, and Aravind Srinivasan. Splitters and near-optimal derandomization. In *36th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 182–191, 1995.

35 Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):52, 2018.

36 Jeanette P Schmidt and Alan Siegel. The spatial complexity of oblivious k-probe hash functions. *SIAM Journal on Computing*, 19(5):775–786, 1990.

The Fine-Grained Complexity of Multi-Dimensional Ordering Properties

Haozhe An
University of Maryland, College Park, MD, USA

Mohit Gurumukhani
University of California at San Diego, CA, USA

Russell Impagliazzo
University of California at San Diego, CA, USA

Michael Jaber
University of California at San Diego, CA, USA

Marvin Künnemann
Institute for Theoretical Studies, ETH Zürich, Switzerland

Maria Paula Parga Nina
University of California at San Diego, CA, USA

Abstract

We define a class of problems whose input is an n-sized set of d-dimensional vectors, and where the problem is first-order definable using comparisons between coordinates. This class captures a wide variety of tasks, such as complex types of orthogonal range search, model-checking first-order properties on geometric intersection graphs, and elementary questions on multidimensional data like verifying Pareto optimality of a choice of data points.

Focusing on constant dimension d, we show that any k-quantifier, d-dimensional such problem is solvable in $O(n^{k-1}\log^{d-1} n)$ time. Furthermore, this algorithm is conditionally tight up to subpolynomial factors: we show that assuming the 3-uniform hyperclique hypothesis, there is a k-quantifier, $(3k-3)$-dimensional problem in this class that requires time $\Omega(n^{k-1-o(1)})$.

Towards identifying a single representative problem for this class, we study the existence of complete problems for the 3-quantifier setting (since 2-quantifier problems can already be solved in near-linear time $O(n\log^{d-1} n)$, and k-quantifier problems with $k > 3$ reduce to the 3-quantifier case). We define a problem Vector Concatenated Non-Domination VCND_d (Given three sets of vectors X, Y and Z of dimension d, d and $2d$, respectively, is there an $x \in X$ and a $y \in Y$ so that their concatenation $x \circ y$ is not dominated by any $z \in Z$, where vector u is dominated by vector v if $u_i \leq v_i$ for each coordinate $1 \leq i \leq d$), and determine it as the "unique" candidate to be complete for this class (under fine-grained assumptions).

2012 ACM Subject Classification Theory of computation → Complexity theory and logic; Theory of computation → Problems, reductions and completeness; Theory of computation → Verification by model checking

Keywords and phrases Fine-grained complexity, First-order logic, Orthogonal vectors

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.3

Funding *Russell Impagliazzo*: Work supported by the Simons Foundation and NSF grant CCF-1909634.
Marvin Künnemann: Research supported by Dr. Max Rössler, by the Walter Haefner Foundation, and by the ETH Zürich Foundation. Part of this research was performed while the author was employed at MPI Informatics.

Acknowledgements We would like to thank Rex Lei, Jiawei Gao, and Victor Vianu for helpful comments and discussion.

1 Introduction

Algorithmic problems based on comparing elements according to a total ordering relation are as fundamental as they are useful. Any introductory algorithms textbook starts with sorting and other comparison-based problems. For higher dimensional data, problems involving comparisons for multiple components, such as range queries, are equally fundamental in computational geometry. In databases, queries need to handle data with many fields that can be compared (beyond other relations on the data), such as listing all employees who are not managers of another employee, with seniority in one range and salary in another.

In this paper, we give a general, systematic study of the complexity of multi-dimensional comparison problems. We define complexity classes capturing the notion of "multi-dimensional comparison problems", as appropriate in geometry and in databases, with the classes PTO_d representing geometric problems in d dimensional data, and TO_d representing problems that combine ordering and other relations for such data, as would be found in databases. We then identify the maximum complexity of problems in these classes under standard assumptions in fine-grained complexity, and relate the classes to each other and other studied complexity classes. For many subclasses, we find natural complete or hard problems where progress on better algorithms for these problems would result in better algorithms for the entire subclass.

While our results are varied, with upper bounds, conditional lower bounds and completeness results, a consistent theme emerges. Our classes are intermediate between two previously studied classes of logically defined problems, first-order in the sparse representation (e.g., graph problems in adjacency list format) and first-order in the dense representation (e.g, graph problems in adjacency matrix format). While orderings are dense relations, with quadratically many pairs for which they hold, they are a special case that can be represented succinctly, by giving an array of ranks for each element. What emerges in our results is that multi-dimensional ordering problems are very tightly connected to first-order in the sparse representation, and not directly connected to the dense representation. Thus, while they give substantially different settings, we give many senses in which sparse relations can be coded in terms of orders, and where orderings can be reduced to sparse relations.

1.1 A class of geometric ordering problems: $PTO_{k,d}$

As an example for multi-dimensional comparison problems, consider 2D orthogonal range searching: given a set of 2-dimensional data points D, answer Boolean queries of the form

$$\exists x \in D : x \in [\ell_1, u_1] \times [\ell_2, u_2],$$

where $[\ell_1, u_1] \times [\ell_2, u_2]$ is a given *orthogonal range*. Note that here, we may without loss of generality replace each point's coordinate in dimension d by its *rank* among the coordinates in dimension d of all points in D. Typical variants include to report, count or optimize over all elements in the query range. A long line of research starting in the 70s, including [37, 42, 22, 11, 40, 19], gives fast algorithms for such tasks, e.g., an algorithm to preprocess D such as to answer queries in time $O(\log \log n)$ using space $O(n \log \log n)$, see [19]. Many more complex algorithmic tasks can be solved using orthogonal range techniques, see [25, 8] for an overview.

Also more complex tasks than mere orthogonal range searching arise naturally: In a set of d-dimensional data points D, consider a *feature* (or property) F of the data points that can be described as being contained in an orthogonal range $[\ell_1, u_1] \times \cdots \times [\ell_d, u_d]$. Given a family $\mathcal{F}$ of such features, there are several natural questions to ask:

- decide if all features are present in the dataset:
  $\forall F = [\ell_1, u_1] \times \cdots \times [\ell_d, u_d] \in \mathcal{F}\ \ \exists x \in D :\ \ x \in F$
- decide if some data point displays all features:
  $\exists x \in D\ \ \forall F = [\ell_1, u_1] \times \cdots \times [\ell_d, u_d] \in \mathcal{F} :\ \ x \in F$
- decide if two different features are equivalent on D:
  $\exists F_1 \in \mathcal{F}\ \ \exists F_2 \in \mathcal{F}\ \ \forall x \in D :\ \ F_1 \neq F_2 \wedge (x \in F_1 \leftrightarrow x \in F_2)$.

Some of these questions can be quickly answered using orthogonal range reporting queries, for others it seems that already the output size of single such query might pose a possibly unnecessary bottleneck. Furthermore, some features might be comparison-based, but more complex than a simple orthogonal range, e.g.,[1]

$$x \in F(\ell_1, u_1, \ldots, \ell_d, u_d) \iff$$
$$(x_1 \in [\ell_1, u_1] \to (x_2 \in [\ell_2, u_2])) \wedge (x_1 \notin [\ell_1, u_1] \to (x_3, \ldots, x_d) \in [\ell_3, u_3] \times \cdots \times [\ell_d, u_d])).$$

In such cases, it would not be immediate whether orthogonal range search techniques can be used at all.

We formalize a notion of "multi-dimensional comparison problems" by introducing a class of problems $PTO_{k,d}$ (for "purely total ordering property") of model-checking a k-quantifier first-order property on a relational structure with d total ordering relations (each succinctly represented as a sorted list of objects) as well as unary relations (to enable comparison of coordinates with constants). In particular, this class contains any property ψ of the form

$$\psi = Q_1 x^{(1)} Q_2 x^{(2)} \ldots Q_k x^{(k)} : \phi(x^{(1)}, \ldots, x^{(k)}),$$

where $Q_i \in \{\exists, \forall\}$, $x^{(i)}$ ranges over a set of d-dimensional vectors (which we also call *objects*), and ϕ is an arbitrary Boolean formula involving only comparisons of the form $x_i^{(a)} \leq x_i^{(b)}$ with $1 \leq a, b \leq k$ (here, $x_i^{(a)}, x_i^{(b)}$ denotes the i-th dimension of $x^{(a)}, x^{(b)}$, respectively), as well as comparisons with constants. We will refer to d as the *dimension* of a formula $\psi \in PTO_{k,d}$. For this paper throughout, we think of ϕ as fixed formula, and thus k, d are constants. See Section 2 for further details.

The class $PTO_{k,d}$ includes all problems as mentioned above, but also tasks such as verifying Pareto optimality of a given set of d-dimensional data points with respect to a superset, or given a set of d-dimensional geometric objects, determine whether there are k distinct such objects whose bounding boxes intersect.

We furthermore extend this class to $TO_{k,d}$, where we allow, beyond d total ordering relations, also arbitrary additional relations (represented explicitly). These two classes encompass in particular the following types of problems:

- Model-checking first order properties of **geometric intersection graphs**: Presence of an edge in an intersection graph of axis-parallel boxes can be decided using comparisons of coordinates. Thus, any k-quantifier first-order property on such geometric intersection graphs in $\mathbb{R}^d$ can be formulated as a problem in $PTO_{k,d}$, such as finding k pair-wise non-intersecting d-dimensional axis-parallel unit-cubes [38].[2]
- **temporal logic**: using a single total ordering relation, we may represent *precedence* in a time domain. Thus, we may express temporal logical statements involving expressions over future or past events in $TO_{k,1}$.

[1] The given expression could model the following feature: if a person is of working age ($x_1 \in [\ell_1, u_1]$), use criterion $x_2 \in [\ell_2, u_2]$, otherwise use $(x_3, \ldots, x_d) \in [\ell_3, u_3] \times \cdots \times [\ell_d, u_d]$.

[2] For even more involved types of algorithmic tasks beyond k-quantifier first-order properties, see, e.g., [20] (All-Pairs Shortest Paths) or [24] (NP-hard problems).

- relational databases with **ordered types**: in relational databases, we may use totally ordered data types (salaries of employees, time events, rank in a sorted list, etc.) as succinct representation to enable comparisons. In this context, studying the complexity of a problem in $TO_{k,d}$ corresponds to studying the *data complexity* of a fixed query.

1.2 Our results

Let $k \geq 2$. We show that any problem in $PTO_{k,d}$ involving n objects can be solved in time $O(n^{k-1}\log^{d-1} n)$ which is $\tilde{O}(n^{k-1})$ for any constant dimension d. We extend this algorithm to run in time $O(m^{k-1}\log^{d-1} m)$ for sentences in $TO_{k,d}$, where m denotes the sum of the number of objects and the size of the additional relations, i.e., the number of tuples contained in the relation. We show the matching conditional lower bound that there is some sentence in $PTO_{k,3k-3}$ that requires time $\Omega(n^{k-1-o(1)})$ under the 3-uniform hyperclique hypothesis [36, 2, 15, 34] – this hypothesis postulates that $n^{k\pm o(1)}$ running time is essentially best possible for finding cliques in hypergraphs. (See the full version of this paper for further details.)

Beyond these general upper and lower bounds, we also seek to identify *hard* or even *complete* problems for this class. Such problems capture the full generality of these classes, in the sense that finding a significantly improved algorithm for this problem would give an improved algorithm for all problems in the class. We use the following fine-grained notion of hardness/completeness: Formally, let P be a problem whose best known algorithm runs in time $T_P(n)$ and let C be a class of problems whose best known algorithms runs in time $T_C(n)$. We say P is *hard* for a class of problems C, if any $T_P(n)^{1-\epsilon}$-time algorithm for P with $\epsilon > 0$ gives a $T_C(n)^{1-\epsilon'}$-time algorithm for all problems in C for some $\epsilon' > 0$. We say that P is *complete* for C, if it is hard for C and contained in C. In particular, if P is complete for C, then P admits substantial improvements over time $T_P(n)$ if and only if *all* problems in C admit substantial improvements over $T_C(n)$.

We identify such problems for specific quantifier structures. In particular, we focus on the 3-quantifier case, since all 2-quantifier $O(1)$-dimensional total order properties can be solved in near-linear time $\tilde{O}(n)$ (Theorem 2), and all k-quantifier properties with $k > 3$ can be reduced to the 3-quantifier case via brute forcing (Corollary 4). Focusing on $PTO_{k,d}$, we obtain the following results (see Table 1):

1. For existentially quantified pure total ordering properties (denoted by $PTO_{\exists\exists\exists,d}$), we give an $\tilde{O}(n^{2\omega/(\omega+1)}) = \tilde{O}(n^{1.407})$ time algorithm and identify the well-studied triangle detection in sparse graphs as a complete problem[3].
2. For the quantifier structure $\forall\exists\exists$, we also give an $\tilde{O}(n^{2\omega/(\omega+1)}) = \tilde{O}(n^{1.407})$ time algorithm by showing that the problem of *counting*, for each edge in a sparse graph, the number of triangles containing this edge is *hard* for the class $PTO_{\forall\exists\exists,d}$. Since we reduce to a counting problem rather than a member of this class, we do not obtain a completeness result, however.
3. For the quantifier structure $\exists\forall\exists$, we were unable to find a complete or hard problem. However, we give evidence that this quantifier structure does not contain a complete problem for $PTO_{k,d}$ by showing that all $PTO_{\exists\forall\exists,d}$ problems have a $\tilde{O}(n)$-time nondeterministic and co-nondeterministic algorithm. Since we also show a $n^{2-o(1)}$ SETH[4]-based

[3] Strictly speaking, we identify the following 3-dimensional problem (which is linear-time equivalent to triangle detection in sparse graphs) as complete for $PTO_{\exists\exists\exists,d}$: $\exists x, y, z : x_1 = z_1 \wedge x_2 = y_2 \wedge y_3 = z_3$.

[4] Strong Exponential Time Hypothesis (SETH) for CNF-SAT: For all $\epsilon > 0$, there exists a k so that k-CNF-SAT cannot be solved in time $O(2^{n(1-\epsilon)})$ [33].

lower bound for $PTO_{3,d}$ when $d \to \infty$, this rules out existence of such a complete problem using deterministic reductions under NSETH, a nondeterministic variant of SETH [17]. We also give a conditional lower bound of $n^{2-o(1)}$ under the Hitting Set conjecture.

4. Finally, for the seemingly most difficult quantifier structure of $\exists\exists\forall$, we show $n^{2-o(1)}$-time conditional lower bounds under SETH and the 3-uniform hyperclique hypothesis, and identify the following complete problem for $PTO_{\exists\exists\forall,d}$, which we call Vector Concatenated Non-Domination VCND_d: Given three sets of vectors X, Y and Z of dimension d, d and $2d$, respectively, is there an $x \in X$ and a $y \in Y$ so that their concatenation $x \circ y$ is not dominated by any $z \in Z$, where vector u is dominated by vector v if $u_i \leq v_i$ for each coordinate $1 \leq i \leq d$.

Note that this covers all quantifier structures for $k = 3$, as deciding $Q_1Q_2Q_3\phi$ with $Q_i \in \{\exists, \forall\}$ is equivalent to deciding $\overline{Q_1}\,\overline{Q_2}\,\overline{Q_3}\,\overline{\phi}$ where $\overline{\forall} = \exists, \overline{\exists} = \forall$ and $\overline{\phi}$ is the negation of ϕ.

These results identify the VCND_d problem as the essentially *only* candidate (up to fine-grained equivalence) to be complete for $PTO_{3,d}$ under NSETH: It is complete for $\exists\exists\forall$, and all problems with a different 3-quantifier structure have either improved deterministic or (co-)nondeterministic algorithms, and thus cannot be complete without major consequences in fine-grained complexity. It remains a challenge to prove or disprove completeness of VCND_d for $PTO_{3,d}$ (beyond its completeness for $PTO_{\exists\exists\forall,d}$).

Since the above results motivate VCND_d as a central problem for $PTO_{k,d}$, we work towards algorithmic improvements for this problem. In particular, we obtain an $\tilde{O}(n^{2-\frac{1}{2^d}})$-time algorithm for VCND whenever one set of vectors is of dimension 2 and the other is of dimension d. Note that obtaining such an $O(n^{2-\epsilon(d)})$ time algorithm with $\epsilon(d) > 0$ for general VCND_d would refute the 3-uniform hyperclique hypothesis by our conditional lower bound and completeness result.

Finally, we show that our algorithmic results extend to the class $TO_{k,d}$ (see Section 3 for details), while all hardness results trivially apply, since they are already proven for the subclass $PTO_{k,d}$. Generally speaking, this shows that the database setting (with additional sparse relations) does not increase the fine-grained complexity compared to the geometric setting of purely total ordering properties.

1.3 Previous work

This work continues a relatively new direction, fine-grained complexity of complexity classes. *Fine-grained complexity* aims to not only qualitatively classify problems as "easy" or "hard", but (to the extent possible) pin-point their exact complexities. We now have a wide collection of standard algorithmic problems where any significant improvements in algorithmic running time would refute one or more conjectures about well-studied problems, such as the k-SUM problem [26], All Pairs Shortest Paths [44, 3, 36], SAT [33, 41], or Orthogonal Vectors [6, 14, 1, 12, 16, 39, 35, 9, 2, 13]. Recent work in fine-grained complexity has gone from considering problems one at a time to following traditional complexity in considering *classes* of problems. Fine-grained reductions often cut across the usual complexity classes (with reductions from NP-complete problems to first-order properties, for example), but on the other hand, fine-grained complexity distinguishes between problems with the same traditional complexities (e.g., two different NP-complete problems might have very different properties in fine-grained complexity). Nevertheless, there are now a number of classes of problems, grouped by logical structure or common format, whose fine-grained complexity is at least partially understood: dense first-order properties [43]; sparse first order properties [17, 29, 15]; several extensions of first order [28]; and certain formats of dynamic programming problems [35, 27].

The most closely related previous work to our results are [43, 29]. Both of these papers consider the class of *first-order definable properties*, the first for the dense case (where each relation is given as a matrix, aka adjacency matrix format), and the second for the sparse case (where the input is given as a list of tuples in the relations, e.g., for graphs, adjacency list format). This class is natural both in terms of computational complexity, where it is the uniform version of AC_0 ([30]), and in database theory, because these are the queries expressible in basic SQL [7]. First-order logic can also express many polynomial time computable problems: Orthogonal Vectors, k-Orthogonal Vectors, k-Clique, k-Independent Set, k-Dominating Set, etc. Not only were the likely complexities of the hardest problems (as a function of number of quantifiers) given, but in the second paper, a natural complete problem was identified, the Orthogonal Vectors problem (OV). The conclusion was that there were substantial improvements possible in the worst-case complexity of model checking for first-order properties if and only if the known algorithms for Orthogonal Vectors can be substantially improved. Using a recent sub-polynomial improvement in OV algorithms by [4, 21], they obtained a similar improvement in model checking for every first-order property. [28] extends this work to related logics such as transitive closure logics, first-order logic on totally ordered sets, and first-order logic with function symbols. They show that model checking for first-order logic with a single total ordering is actually equivalent to that for unordered structures under fine-grained reductions. In contrast, we show that for even two orderings, the model checking problem becomes substantially harder, meaning we require new techniques to characterize the complexity of problems on multi-dimensional data.

There is also work on classes of problems that are related in spirit, but do not form a well-studied complexity class. V.-Williams and Williams [44] study problems related to shortest paths in graphs, and shows that many are subcubic-time equivalent. Künnemann et al. [35] study dynamic programming problems with a similar structure and give a unified treatment of their fine-grained complexities. Gao [27] extends this class of dynamic programming problems from lines to tree-like structures such as bounded treewidth graphs.

2 Preliminaries

The following notion of *fine-grained reductions* was introduced in [44].

▶ **Definition 1** (Fine-grained reduction). *Let* $(\Pi_1, T_1(m)) \leq_{FGR} (\Pi_2, T_2(m))$ *denote that for every* $\epsilon > 0$ *there is a* $\delta > 0$ *and a Turing reduction from* Π_1 *to* Π_2 *so that the time for the reduction (not counting oracle calls) is* $O(T_1(m)^{1-\delta})$ *and* $\sum_q (T_2(|q|))^{1-\epsilon} \in O(T_1(m)^{1-\delta})$*, where the sum is over all oracle calls* q *made by the reduction on an instance of size* m*.*

In other words, if there is some $\epsilon > 0$ such that problem Π_2 is in $\mathsf{TIME}((T_2(m))^{1-\epsilon})$, then problem Π_1 is in $\mathsf{TIME}((T_1(m))^{1-\delta})$ for some $\delta > 0$, i.e., if Π_2 can be solved substantially faster than T_2 then Π_1 can be solved substantially faster than T_1. If both T_1 and T_2 are $\Theta(m^2)$, the reduction is called a subquadratic reduction. We say that Π_1 and Π_2 are *fine-grained equivalent* if there is a fine-grained reduction from Π_1 to Π_2 and vice versa.

We use this notation not only on single problems but also on classes of problems. Let C_1 and C_2 be classes problems. $(C_1, T_1(m)) \leq_{\text{FGR}} (C_2, T_2(m))$ if for all problems $\Pi_1 \in C_1$ there is a $\Pi_2 \in C_2$ so that $(\Pi_1, T_1(m))$ fine-grained reduces to $(\Pi_2, T_2(m))$.

Details on $PTO_{k,d}$ and $TO_{k,d}$

In this paper, we consider the fine-grained complexity of model checking problems definable in first-order logic on structures with d binary relations $x \leq_i y$, $1 \leq i \leq d$, where each binary relation is a total pre-order of the universe (i.e., transitive, reflective, total, but not necessarily anti-symmetric.)

Total orders. We use $x \leq_i y$ to represent the i'th relation in our family holding between x and y. Such a relation is dense, holding for $\Theta(n^2)$ pairs of elements. However, we can represent such a representation succinctly, by giving an array which for each element specifies its rank in a list sorted by the ordering relation (with some elements having the same rank, if inequality holds in both directions). It is in this succinct format that ordering relations are described for our problems.

Equivalently, we may represent all ordering relations by representing each object x as a d-dimensional vector $(x_1, \ldots, x_d)$, where x_i denotes the rank of x in the i'th ordering relation. Thus, it is equivalent to write $x \leq_i y$ or $x_i \leq y_i$, and we will switch between these two based on which seems clearer for the given circumstance.

The vectors we get in this way are very special, in that the coordinates are always positive integers from 1 to n. However, also problems defined about d dimensional vectors over any totally ordered domain (such as $\mathbb{R}$) fall into our setting, since in $O(n \log n)$ time we can replace each x_i with its rank in the set of i'th coordinates of vectors.

Unary relations. We also allow unary relations, or, equivalently, comparisons to constants. More precisely, any unary relation U is represented as a list of objects for which U holds. Apart from allowing us to put objects into categories (sometimes called *colored* properties), this enables us to express comparisons of coordinates with constants: To express whether $x \leq_i \gamma$ for some constant γ, we introduce a unary relation symbol $U_i^{\leq \gamma}$ that holds for all x with $x_i \leq \gamma$. Thus from now on, it suffices to declare constants γ explicitly, and afterwards we may express arbitrary comparisons like $x_i \neq \gamma$ or $x_i > \gamma$. Note that since we always consider fixed formulas ψ, each considered property will use $O(1)$ constants for comparisons.

Definition of $PTO_{k,d}$. We denote the class of purely total ordering model-checking problems for first-order formulas in pre-orderings and unary relations specified as above where the formula has d distinct ordering relations and k total occurrences of quantifiers by $PTO_{k,d}$. PTO_k is the union of $PTO_{k,d}$ over all constants d. We can further divide PTO_k into 2^k sub-classes based on the quantifier structure, so for example $PTO_{\exists\exists\exists}$ is the sub-class of PTO_3 where the model-checking problems are for formulas of the form $\exists x \exists y \exists z \Phi(x, y, z)$ where Φ is quantifier-free. We let n be the size of the universe of the structure, which is also, up to constant factors, the size in terms of $O(\log n)$-bit words required to specify all total pre-orderings and unary relations. Algorithm time for problems in PTO is thus measured in terms of n. In this format, it is a constant time operation to evaluate whether any relation is true or false for specified elements.

Definition of $TO_{k,d}$. We generalize $PTO_{k,d}$ to the class $TO_{k,d}$ by also allowing the formula and models to have any constant number of sparse relations of any constant arity. These are specified as lists of tuples where the relation holds. Let the problem size be denoted by m, which is equal to the sum of the number of elements n and the number of tuples.

We assume all algorithms start with quasi-linear time preprocessing steps to create data structures such as hash tables or binary search trees that allow fast determination (constant time or logarithmic time) of whether a relation holds for given elements, and allows one to list the tuples in a relation that contain a given element in at most poly-log time + poly-log time times the number of such tuples.

On the difference. $PTO_{k,d}$ is a more "geometric" class of problems, and so it is interesting when we can reduce combinatorial problems to this class. Therefore, we will focus on these classes when giving conditional hardness results. $TO_{k,d}$ is closer to the type of problems

Table 1 Our results for $PTO_{k,d}$, where we assume that d is an arbitrarily large constant.

Quantifier structure	3 quantifiers		k quantifiers, $k > 3$	
$\ldots\exists\exists\forall$ (sym.: $\ldots\forall\forall\exists$) complete: VCND_d (Thm. 13)	$\tilde{O}(n^2)$	$n^{2-o(1)}$ for $d = 6$ (3-unif. HC, Thm. 14) $n^{2-o(1)}$ for $d \to \infty$ (SETH, Thm. 15)	$\tilde{O}(n^{k-1})$	$n^{k-1-o(1)}$ for $d = 3k-3$ (3-unif. HC, Thm. 14)
$\ldots\exists\forall\exists$ (sym.: $\ldots\forall\exists\forall$) complete: open	$\tilde{O}(n^2)$ $\tilde{O}(n)$ (co-)nondet.	$n^{2-o(1)}$ for $d \to \infty$ (Hitting Set, Thm. 11)	$\tilde{O}(n^{k-1})$ $\tilde{O}(n^{k-2})$ (co-)nondet.	 $n^{k-2-o(1)}$ for $d = 2$ (SETH, Thm. 12)
$\ldots\forall\exists\exists$ (sym.: $\ldots\exists\forall\forall$) complete: open hard: ETC (Thm. 8)	$\tilde{O}(n^{\frac{2\omega}{\omega+1}})$ $= O(n^{1.41})$		$\tilde{O}(n^{k-\frac{\omega+3}{\omega+1}})$ $= O(n^{k-1.59})$	
$\ldots\exists\exists\exists$ (sym.: $\ldots\forall\forall\forall$) complete: triangle det. (Thm. 6)	$\tilde{O}(n^{\frac{2\omega}{\omega+1}})$ $= O(n^{1.41})$		$\tilde{O}(n^{k-\frac{\omega+3}{\omega+1}})$ $= O(n^{k-1.59})$	

that might arise in applications such as database queries. Therefore, we will focus on $TO_{k,d}$ when giving algorithms or other upper bounds on complexity. Since $PTO_{k,d} \subseteq TO_{k,d}$, lower bounds for $PTO_{k,d}$ are stronger results, and upper bounds for $TO_{k,d}$ are stronger results.

Further examples of problems in $PTO_{k,d}$. To define further well-studied problems in $PTO_{k,d}$, we say that a vector u *dominates* vector v if $u_i \geq v_i$ for all $1 \leq i \leq d$, and denote this by $u \geq_{dom} v$. Furthermore, given a set of d-dimensional real vectors A, we say that vector set B is *Pareto-optimal for* A if for every $a \in A$ there is a $b \in B$ with $b \geq_{dom} a$.

- Vector Domination Problem (see, e.g. [31]): Given two sets of d-dimensional real vectors A and B, are there two vectors $u \in A$ and $v \in B$ such that $u \geq_{dom} v$?
- Pareto Optimality Verification (see, e.g. [32]): Given a set A of vectors, and a candidate vector set B, determine whether B is indeed Pareto optimal for A.

From the definition, both problems are in $PTO_{2,d}$. As we will see, they can be solved in time $O(n \log^{d-1} n)$. For superconstant dimension d, [31, 18] give further improvements.

3 Technical Overview

In this section, we give the main ideas for all of our results, see Table 1 for an overview. Due to space constraints, the proofs are deferred to the full version of this paper.

One of our main results is an upper-bound on model-checking sentences in $PTO_{k,d}$ and $TO_{k,d}$.

▶ **Theorem 2.** *There is an algorithm running in time $O(n \log^{d-1}(n))$ for model-checking a two-quantifier formula $Q_1 x Q_2 y \varphi(x, y)$ with d ordering relations and unary predicates.*

Specifically, we obtain this result using the following lemma, which we obtain by a reduction to orthogonal range counting.

▶ **Lemma 3.** *Given a formula $\varphi(x, y)$ with d ordering relations and unary predicates and two sets X, Y of vectors in $\mathbb{R}^d$, there is an $O(n \log^{d-1}(n))$ time algorithm that returns an array A indexed by each $x \in X$ so that $A[x]$ is the number of $y \in Y$ so that $\varphi(x, y)$ is true.*

Combining the above theorem with exhaustive search over the first $k-2$ quantifiers yields

▶ **Corollary 4.** *Model-checking formulas in $PTO_{k,d}$ is in* $\mathsf{TIME}(n^{k-1}\log^{d-1}(n))$.

If we have additional explicitly represented relations, more work is required. For such cases, throughout the paper, we will always assume that these relations are *sparse*, i.e., the total input size is $m = O(n)$. In this case, we obtain the same asymptotic running time.

▶ **Theorem 5.** *Model-checking formulas in $TO_{k,d}$ is in* $\mathsf{TIME}(m^{k-1}\log^{d-1}(m))$.

The idea is to reduce the problem to the purely totally ordered case by assuming that all sparse relations are empty; using Lemma 3 for the 2-quantifier case, we can obtain for each x the number of y satisfying the condition. We then repair these counts to the true values by iterating over the additional sparse relations, similar to the baseline algorithm in [29].

Note that in Section 3.4, we discuss a lower bound proving these baseline algorithms to be conditionally optimal under fine-grained hardness assumptions.

In the remainder of the section, we distinguish our results based on the quantifier structure. Since any k-quantifier formula with $k > 3$ reduces to the 3-quantifier setting via brute force over the first $k-3$ quantifiers, we only regard 3-quantifier structures.

3.1 Quantifier Structures Ending in ∃∃∃

Recall that informally, we call a problem *complete* for a class if it is contained in the class and model-checking any sentence in the class reduces to our problem. For sentences in $PTO_{k,d}$ ending in ∃∃∃, we show that detecting triangles in a sparse graph is complete for this class. By current running time bounds for the problem [10], we obtain a running time of $\tilde{O}(n^{2\omega/(\omega+1)}) = \tilde{O}(n^{1.407\ldots})$.

▶ **Theorem 6.** *The triangle detection problem in sparse graphs is fine-grained equivalent to a problem that is complete for model-checking ∃∃∃ formulas with only ordering relations and unary relations.*

More precisely, the following ordering property is shown to be complete: $\exists x \exists y \exists z : x_1 = z_1 \wedge x_2 = y_2 \wedge y_3 = z_3$ which is easy to be seen equivalent to triangle detection in sparse graphs.

Intuitively, we reduce to this problem as follows: Given a formula $\exists x \exists y \exists z \phi(x, y, z)$, we can determine whether $\phi(x, y, z)$ holds once we know all comparisons between x, y, z in each dimension i. A challenge here is to reduce comparisons like $x_i < y_i$ to an equality check: Similar to a trick used in [45], we do this by guessing the highest-order bit of *divergence* between x_i and y_i to obtain a "proof" only involving equalities; since we may assume that $1 \leq x_i, y_i \leq n$ (by working in *rank* space), there are only $O(\log n)$ choices for a single comparison. The key observation is that the quantifier structure is sufficiently well behaved to make this reduction work: we only need to guess these bits of divergence for $O(d)$ many comparisons and can express correctness of all proofs for comparisons between x and z using equality on the first dimension, between x and y using the second dimension, and between y and z using the third dimension. In total, this results in an admissible blow-up of $\log^{O(d)} n$.

We turn to the setting with additional sparse relations, i.e., formulas in $TO_{\exists\exists\exists,d}$. Here we establish the triangle *counting* problem in sparse graphs as hard for the class. Since the approach of [10] also gives a counting algorithm in the same running time as detection, we establish the same algorithmic upper bound.

▶ **Theorem 7.** *Every problem in $TO_{\exists\exists\exists,d}$ reduces to the problem of counting the number of triangles in a sparse graph via reductions that preserve time up to polylog factors.*

Handling the additional sparse relations is highly non-trivial. In particular, to obtain our result, we first show that the triangle counting problem is hard for model-counting $\exists\exists\exists$ formulas in the sparse setting of [29], which is interesting in its own right.

Since triangle detection is a classical problem, improving the bound of $O(n^{1.407})$ for $\exists\exists\exists$ structures already in the purely total ordering case would be a major algorithmic result.

3.2 Quantifier Structures Ending in $\forall\exists\exists$

For quantifier structures ending in $\forall\exists\exists$, we obtain a hard problem: We show that every problem in $TO_{\forall\exists\exists,d}$ (and thus also $PTO_{\forall\exists\exists,d}$) reduces to that of determining, for each edge in a sparse graph, how many triangles contain this edge; we call this problem *Edgewise Triangle Counting (ETC)*. Again, currently the best algorithm for this problem is essentially the same as that for triangle detection and counting [10].

▶ **Theorem 8.** *Edgewise Triangle Counting is hard for model-checking $TO_{\forall\exists\exists,d}$ formulas.*

Since the high-level arguments for this results substantially build on the hardness result for $TO_{\exists\exists\exists,d}$, we defer all details for this result to the full version of this paper.

3.3 Quantifier Structures ending in $\exists\forall\exists$

For the quantifier structure of $\exists\forall\exists$, we are unable to establish a complete problem. However, this quantifier structure admits (co-)nondeterministic algorithms that are faster than the baseline algorithm.

▶ **Theorem 9.** *Model-checking formulas in $PTO_{k,d}$ ending in $\exists\forall\exists$ can be done in nondeterministic and co-nondeterministic time $O(n^{k-2}\log^{d-1}(n))$.*

The main idea is as follows: Consider any $\exists x\forall y Q z\phi(x,y,z)$ property. For the nondeterministic algorithm, we simply (nondeterministically) guess x and solve the remaining 2-quantifier problem $\forall y Q z\phi(x,y,z)$ in time $O(n\log^{d-1} n)$ using the baseline algorithm. For the co-nondeterministic algorithm, we need to verify that $\forall x\exists y\overline{Q}z\overline{\phi}(x,y,z)$. Here, for every x, we (nondeterministically) guess a witness y_x and solve the remaining $\overline{Q}z\overline{\phi}(x,y_x,z)$ formula using the approach of Theorem 2.

For the case of total ordering properties with additional sparse relations, this approach is not directly applicable: If, e.g., all guessed witnesses y_x happen to participate in many tuples of the sparse relations, we have to repeatedly solve problems with a large input size. We remedy this problem by taking care of such large degree witness y_x explicitly; while this incurs a certain slow-down, we can limit it to a factor of $O(\sqrt{n})$.

▶ **Theorem 10.** *Model-checking formulas in $TO_{k,d}$ ending in $\exists\forall\exists$ can be done in nondeterministic and co-nondeterministic time $O(m^{k-3/2}\log^{d-1}(m))$.*

As a consequence of the above nondeterministic algorithms, assuming NSETH [17], we cannot establish hardness beyond $n^{k-2-o(1)}$ for $PTO_{\exists\forall\exists,d}$ using deterministic SETH-based reductions. However, by reducing from a problem with low (co-)nondeterministic complexity, specifically, the Hitting Set conjecture [5], we can give a conditional lower bound already for $PTO_{\exists\forall\exists,d}$ (as $d\to\infty$) that matches our baseline algorithm.

▶ **Theorem 11.** *Assuming the Hitting Set conjecture, for all $\epsilon > 0$, there exists some d and a $PTO_{\exists\forall\exists,d}$ sentence that cannot be solved in time $O(n^{2-\epsilon})$.*

The proof of this result is reminiscent to some reductions in [23]. We reduce from Hitting Set (given sets of vectors $A, B \subseteq \{0,1\}^{c \log n}$ for arbitrary c, determine whether some $a \in A$ is non-orthogonal to all $b \in B$) to a formula $\exists x \forall y \exists z \psi(x, y, z)$ as follows: We think of x ranging over vectors $a \in A$, y ranging over $b \in B$, and think of z as a "proof" of the fact that a, b are non-orthogonal, given by a prover Merlin. There is a trade-off between size of the proofs and the required dimension to represent the vectors, which we set in a way that bounds the number of possible proofs to $O(n)$, resulting in a dimension d growing only with c (independently of n).

We also give a conditional lower bound from SETH for $k > 3$ that matches the NSETH barrier following from the (co-)nondeterministic algorithms. Notably, this lower bound already applies to dimension $d = 2$.

▶ **Theorem 12.** *Assuming SETH, there exists some $PTO_{k,2}$ sentence ending in $\exists\forall\exists$ that cannot be solved in time $O(n^{k-2-\epsilon})$ for any $\epsilon > 0$.*

We reduce the k-Orthogonal Vectors problem into an $\exists^k\forall\exists$-quantified 2-dimensional formula. Intuitively, the first k existential quantifiers choose k vectors, the $\forall$-quantifier ranges over all vector-dimensions to test, and crucially, the final $\exists$-quantifier enables to guess which of the k vectors has a 0-coordinate in this vector-dimension. Here, the final $\exists$-quantifier is instrumental in making the formula's dimension independent of the vector dimensions.

3.4 Quantifier Structures ending in $\exists\exists\forall$

For sentences in $PTO_{k,d}$ ending in $\exists\exists\forall$, we obtain the complete problem VCND_d: Given three sets of vectors X, Y and Z of dimension d, d and $2d$, respectively, determine if there an $x \in X$ and a $y \in Y$ so that their concatenation $x \circ y$ is not dominated by any $z \in Z$.

▶ **Theorem 13.** *For all d, there exists a d' such that $\mathsf{VCND}_{d'}$ is complete for model-checking $\exists\exists\forall$ formulas in $PTO_{k,d}$.*

This is one of our most interesting results. We reduce a formula $\exists x \in X \exists y \in Y \forall z \in Z : \psi(x, y, z)$ to VCND_d as follows: We carefully divide all pairs in $X \times Y$ into instances $(X_1, Y_1), \ldots, (X_L, Y_L)$ such that for each instance (X_ℓ, Y_ℓ), all comparisons $x_i < y_i, x_i = y_i, x_i > y_i$ for all dimensions i are uniform among pairs $x \in X_\ell, y \in Y_\ell$. Thus, for each ℓ, we may simplify ψ to a formula ψ_ℓ not involving comparisons between x and y. In particular, we may express ψ_ℓ in CNF, where each clause is a disjunction of $\{<, \leq, \geq, >\}$-comparisons between x_i and z_i or between y_i and z_i (in some dimension i). Since all such clauses need to be fulfilled simultaneously, for each $z \in Z$ and clause C, we introduce some z_C chosen such that the clause C is falsified if and only if $x \circ y$ are dominated by z_C.

We show a matching conditional lower bound of $n^{k-o(1)}$ for $PTO_{\exists^k\forall,d}$ under the 3-uniform hyperclique hypothesis.

▶ **Theorem 14.** *For $k \geq 2$ and $h \geq 3$, under the h-uniform HyperClique hypothesis, there is a sentence in $PTO_{k+1,hk}$ ending in $\exists\exists\forall$ that requires time $\Omega(n^{k-o(1)})$.*

We use the first k quantifiers to represent a choice of clique nodes, each represented in its own dimension, and use the $\forall$ quantifier to check that no forbidden configuration is used (a non-edge in the given hypergraph). Naively, this would create $\Theta(n^3)$ rather than $O(n)$ objects, which we remedy by reducing from finding hypercliques of size hk (rather than k).

We also establish a SETH-based lower bound directly for VCND_d. The reduction is very similar to our Hitting-Set-based lower bound for $\exists\forall\exists$-structures.

▶ **Theorem 15.** *Assuming SETH, for every $\epsilon > 0$, there is a d such that VCND_d requires time $\Omega(n^{2-\epsilon})$.*

Specialized algorithm for VCND_d. Since our completeness results establishes VCND_d as a central problem for the study of $PTO_{k,d}$, we consider special cases of the problem. In particular, if one of these sets contains vectors of dimension 2, while the other contains vectors of dimension d, we show the following algorithm, which uses the Erdös-Szekeres Theorem as main ingredient. We use this to extract lists of vectors so that when we restrict to any dimension, the vectors appear in monotonic increasing or decreasing order. This way, the vectors that dominate some fixed vector x form an interval, which allows us to take advantage of fast segment trees that solve an interval covering problem.

▶ **Theorem 16.** *There is a $\widetilde{O}(n^{2-\frac{1}{2^d}})$ time algorithm for VCND when one set of vectors is of dimension 2 and the other is of dimension d.*

Note that such an improvement to $\tilde{O}(n^{2-\epsilon(d)})$ with $\epsilon(d) > 0$ for the general VCND_d problem would refute the 3-uniform hyperclique hypothesis by Theorem 14. In the appendix, we also give an algorithm for very high-dimensional VCND_d.

4 Conclusion and open problems

We have introduced general classes $TO_{k,d}, PTO_{k,d}$ of multidimensional ordering problems as model-checking problems for k-quantifier first-order formulas over d succinctly represented ordering relations (with or without additional explicitly represented relations). We gave a conditionally tight algorithm running in time $O(m^{k-1}\log^d m)$ for all these problems. For $PTO_{k,d}$, we gave complete or hard problems for most quantifier structures, and identified a problem VCND_d as the essentially only candidate to be complete for $PTO_{k,d}$.

The main open problem is to prove or disprove that VCND_d is complete for $PTO_{k,d}$. The major challenge here is to reduce $\exists\forall\exists$-quantified ordering problems to the $\exists\exists\forall$-quantified VCND_d. Such a reduction is possible in the unordered setting [29], but its unclear how to make this approach work in our setting. Likewise, can we prove that a hybrid version of VCND_d and the orthogonal vectors problem (which is complete for the sparse-relational setting [29]) is complete for $TO_{k,d}$? An intermediate step could be to find a complete problem for $\exists\forall\exists$-quantified ordering problems.

A further general algorithmic question is to study existence of improved algorithms for very small constant dimensions d, such as $d = 1$ and $d = 2$, in particular the existence of $O(n^{2-\epsilon(d)})$ time algorithms with $\epsilon(d) > 0$, for 3-quantifier problems. In this direction, we have given an $O(n^{2-\frac{1}{2^d}})$-time algorithm for the central VCND problem where one set of vectors has dimension 2 and the other has dimension d. Note that by our results, such an algorithm for the general VCND_d problem would refute the 3-uniform HyperClique conjecture. Can we classify which problems admit such improved algorithms for small dimensions?

References

1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 59–78. IEEE, 2015.

2 Amir Abboud, Karl Bringmann, Holger Dell, and Jesper Nederlof. More consequences of falsifying SETH and the orthogonal vectors conjecture. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 253–266. ACM, 2018.

3 Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, apsp and diameter. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 1681–1697. SIAM, 2014.

4 Amir Abboud, Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 218–230. SIAM, 2015.

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

6 Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *International Colloquium on Automata, Languages, and Programming*, pages 39–51. Springer, 2014.

7 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases: the logical level*. Addison-Wesley Longman Publishing Co., Inc., 1995.

8 Pankaj K. Agarwal. Range searching. In Jacob E. Goodman, Joseph O'Rourke, and Csaba D. Tóth, editors, *Handbook of Discrete and Computational Geometry*, chapter 40. CRC Press LLC, 3rd edition, 2017.

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

10 Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997. doi:10.1007/BF02523189.

11 Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. New data structures for orthogonal range searching. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 198–207. IEEE Computer Society, 2000. doi:10.1109/SFCS.2000.892088.

12 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 51–58. ACM, 2015.

13 Arturs Backurs, Liam Roditty, Gilad Segal, Virginia Vassilevska Williams, and Nicole Wein. Towards tight approximation bounds for graph diameter and eccentricities. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 267–280. ACM, 2018.

14 Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless seth fails. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 661–670. IEEE, 2014.

15 Karl Bringmann, Nick Fischer, and Marvin Künnemann. A fine-grained analogue of schaefer's theorem in P: dichotomy of existsˆk-forall-quantified first-order graph properties. In Amir Shpilka, editor, *34th Computational Complexity Conference, CCC 2019, July 18-20, 2019, New Brunswick, NJ, USA*, volume 137 of *LIPIcs*, pages 31:1–31:27. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CCC.2019.31.

16 Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 79–97. IEEE, 2015.

17 Marco L. Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mihajlin, Ramamohan Paturi, and Stefan Schneider. Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility. In Madhu Sudan, editor, *Proceedings of the 2016 ACM*

Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016, pages 261–270. ACM, 2016. doi:10.1145/2840728.2840746.

18 Timothy M. Chan. Orthogonal range searching in moderate dimensions: k-d trees and range trees strike back. *Discret. Comput. Geom.*, 61(4):899–922, 2019.

19 Timothy M. Chan, Kasper Green Larsen, and Mihai Patrascu. Orthogonal range searching on the ram, revisited. In Ferran Hurtado and Marc J. van Kreveld, editors, *Proceedings of the 27th ACM Symposium on Computational Geometry, Paris, France, June 13-15, 2011*, pages 1–10. ACM, 2011. doi:10.1145/1998196.1998198.

20 Timothy M. Chan and Dimitrios Skrepetos. All-pairs shortest paths in geometric intersection graphs. *JoCG*, 10(1):27–41, 2019. doi:10.20382/jocg.v10i1a2.

21 Timothy M Chan and Ryan Williams. Deterministic APSP, Orthogonal Vectors, and More: Quickly derandomizing Razborov-Smolensky. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1246–1255. SIAM, 2016.

22 Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Comput.*, 17(3):427–462, 1988. doi:10.1137/0217026.

23 Lijie Chen and Ryan Williams. An equivalence class for orthogonal vectors. 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 21–40. SIAM, 2019. doi:10.1137/1.9781611975482.2.

24 Mark de Berg, Hans L. Bodlaender, Sándor Kisfaludi-Bak, Dániel Marx, and Tom C. van der Zanden. A framework for eth-tight algorithms and lower bounds in geometric intersection graphs. In 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 574–586. ACM, 2018. doi:10.1145/3188745.3188854.

25 Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Cheong Schwarzkopf. *Orthogonal Range Searching*, pages 95–120. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000. doi:10.1007/978-3-662-04245-8_5.

26 Anka Gajentaan and Mark H Overmars. On a class of $o(n^2)$ problems in computational geometry. *Computational geometry*, 5(3):165–185, 1995.

27 Jiawei Gao. On the fine-grained complexity of least weight subsequence in multitrees and bounded treewidth dags. In Bart M. P. Jansen and Jan Arne Telle, editors, *14th International Symposium on Parameterized and Exact Computation, IPEC 2019, September 11-13, 2019, Munich, Germany*, volume 148 of *LIPIcs*, pages 16:1–16:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.IPEC.2019.16.

28 Jiawei Gao and Russell Impagliazzo. The fine-grained complexity of strengthenings of first-order logic. *Electronic Colloquium on Computational Complexity (ECCC)*, 26:9, 2019. URL: https://eccc.weizmann.ac.il/report/2019/009.

29 Jiawei Gao, Russell Impagliazzo, Antonina Kolokolova, and Ryan Williams. Completeness for first-order properties on sparse structures with algorithmic applications. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, pages 2162–2181, 2017.

30 Neil Immerman. *Descriptive complexity*. Graduate texts in computer science. Springer, 1999. doi:10.1007/978-1-4612-0539-5.

31 Russell Impagliazzo, Shachar Lovett, Ramamohan Paturi, and Stefan Schneider. 0-1 integer linear programming with a linear number of constraints. *CoRR*, abs/1401.5512, 2014. arXiv:1401.5512.

32 Russell Impagliazzo, Ramamohan Paturi, and Stefan Schneider. A satisfiability algorithm for sparse depth-2 threshold circuits. *CoRR*, abs/1212.4548, 2012. arXiv:1212.4548.

33 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? In *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on*, pages 653–662. IEEE, 1998.

34 Marvin Künnemann and Dániel Marx. Finding small satisfying assignments faster than brute force: A fine-grained perspective into boolean constraint satisfaction. In *Proc. 35th Computational Complexity Conference (CCC 2020)*, 2020. To appear.

35 Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. On the Fine-Grained Complexity of One-Dimensional Dynamic Programming. In *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:15, 2017.

36 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. Society for Industrial and Applied Mathematics, 2018.

37 George S. Lueker. A data structure for orthogonal range queries. In *19th Annual Symposium on Foundations of Computer Science, Ann Arbor, Michigan, USA, 16-18 October 1978*, pages 28–34. IEEE Computer Society, 1978. doi:10.1109/SFCS.1978.1.

38 Dániel Marx and Anastasios Sidiropoulos. The limited blessing of low dimensionality: when 1-1/d is the best possible exponent for d-dimensional geometric problems. In Siu-Wing Cheng and Olivier Devillers, editors, *30th Annual Symposium on Computational Geometry, SOCG'14, Kyoto, Japan, June 08 - 11, 2014*, page 67. ACM, 2014. doi:10.1145/2582112.2582124.

39 Daniel Moeller, Ramamohan Paturi, and Stefan Schneider. Subquadratic algorithms for succinct stable matching. In *International Computer Science Symposium in Russia*, pages 294–308. Springer, 2016.

40 Yakov Nekrich. Orthogonal range searching in linear and almost-linear space. *Comput. Geom.*, 42(4):342–351, 2009. doi:10.1016/j.comgeo.2008.09.001.

41 Mihai Patrascu and Ryan Williams. On the possibility of faster SAT algorithms. In *SODA*, volume 10, pages 1065–1075. SIAM, 2010.

42 D.E. Willard. *Predicate-oriented Database Search Algorithms*. Center for Research in Computing Technology: Center for Research in Computing Technology. Garland Pub., 1979. URL: https://books.google.de/books?id=iLQmAAAAMAAJ.

43 Ryan Williams. Faster decision of first-order graph properties. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, page 80. ACM, 2014.

44 Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 645–654. IEEE, 2010.

45 Virginia Vassilevska Williams and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. *SIAM J. Comput.*, 42(3):831–854, 2013. doi:10.1137/09076619X.

A New Framework for Kernelization Lower Bounds: The Case of Maximum Minimal Vertex Cover

Júlio Araújo
Departament of Mathematics, Federal University of Ceará, Fortaleza, Brazil

Marin Bougeret
LIRMM, Université de Montpellier, France

Victor Campos
Departament of Computer Science, Federal University of Ceará, Fortaleza, Brazil

Ignasi Sau
LIRMM, Université de Montpellier, CNRS, France

Abstract

In the Maximum Minimal Vertex Cover (MMVC) problem, we are given a graph G and a positive integer k, and the objective is to decide whether G contains a minimal vertex cover of size at least k. Motivated by the kernelization of MMVC with parameter k, our main contribution is to introduce a simple general framework to obtain lower bounds on the degrees of a certain type of polynomial kernels for vertex-optimization problems, which we call lop-*kernels*. Informally, this type of kernels is required to preserve large optimal solutions in the reduced instance, and captures the vast majority of existing kernels in the literature. As a consequence of this framework, we show that the trivial quadratic kernel for MMVC is essentially optimal, answering a question of Boria et al. [Discret. Appl. Math. 2015], and that the known cubic kernel for Maximum Minimal Feedback Vertex Set is also essentially optimal. On the positive side, given the (plausible) non-existence of subquadratic kernels for MMVC on general graphs, we provide subquadratic kernels on H-free graphs for several graphs H, such as the bull, the paw, or the complete graphs, by making use of the Erdős-Hajnal property in order to find an appropriate decomposition. Finally, we prove that MMVC does not admit polynomial kernels parameterized by the size of a minimum vertex cover of the input graph, even on bipartite graphs, unless NP $\subseteq$ coNP/poly. This indicates that parameters smaller than the solution size are unlike to yield polynomial kernels for MMVC.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Maximum minimal vertex cover, parameterized complexity, polynomial kernel, kernelization lower bound, Erdős-Hajnal property, induced subgraphs

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.4

Related Version *Full Version*: https://arxiv.org/abs/2102.02484

Funding *Júlio Araújo*: CNPq-Pq 304478/2018-0, CAPES-PrInt 88887.466468/2019-00 and CAPES-STIC-AmSud 88881.569474/2020-01.
Victor Campos: FUNCAP - PNE-011200061.01.00/16.
Ignasi Sau: DEMOGRAPH (ANR-16-CE40-0028), ESIGMA (ANR-17-CE23-0010), ELIT (ANR-20-CE48-0008-01), and French-German Collaboration ANR/DFG Project UTMA (ANR-20-CE92-0027).

Acknowledgements We would like to thank Michael Lampis (resp. Magnus Wahlström, Venkatesh Raman) for pointing us to reference [27] (resp. reference [34], references [7, 44]).

1 Introduction

A *vertex cover* in a graph G is a subset of vertices containing at least one endpoint of every edge. In the associated optimization problem, called Minimum Vertex Cover, the objective is to find, given an input graph G, a vertex cover in G of minimum size. This

problem has been one of the leitmotifs of the area of parameterized complexity [21,26], serving as a test bed for many of the most fundamental techniques. An instance of a *parameterized problem* is of the form (x, k), where x is the total input (typically, a graph) and k is a positive integer called the *parameter*. The crucial notion is that of *fixed-parameter tractable* algorithm, FPT for short, which is an algorithm deciding whether (x, k) is a positive instance in time $f(k) \cdot |x|^{\mathcal{O}(1)}$, where f is a computable function depending only on k. In the parameterized Vertex Cover problem, we are given a graph G and an integer parameter k, and the objective is to decide whether G contains a vertex cover of size at most k. One of the main fields within parameterized complexity is *kernelization* [31], where the objective is to decide whether an instance (x, k) of a parameterized problem can be transformed in polynomial time into an equivalent instance (x', k') whose total size is bounded by a function of k; the reduced instance is called a *kernel*, and finding kernels of small size, typically polynomial or even linear in k in the best case, is one of the most active areas of parameterized complexity. There are several techniques for obtaining linear kernels for the Vertex Cover problem [31].

Considering the "max-min" version of minimization problems, that is, maximizing the size of a *minimal* solution of the corresponding problem, is a natural approach that has been applied to several problems such as Dominating Set [5,28] (whose "max-min" version is called Upper Domination), Feedback Vertex Set [27], or Hitting Set [4,22]. In this article we are interested in the "max-min" version of Minimum Vertex Cover, called Maximum Minimal Vertex Cover, or just MMVC for short.

Previous work. In his habilitation, Fernau [30] presented FPT algorithms for MMVC as well as some results about its kernelization parameterized by the solution size k. It is easy to note, as observed in [30], that the problem admits a kernel with at most k^2 vertices: if some vertex has degree at least k, we can safely answer "yes" (cf. Lemma 2 for a proof); otherwise, the maximum degree is at most $k-1$, and it follows that every instance without isolated vertices (which may be safely removed) that has at least k^2 vertices is a yes-instance, hence we have a trivial kernel with at most k^2 vertices. Fernau [30] presented a kernel with at most $4k$ vertices for MMVC restricted to planar instances using the algorithmic version of the Four Color Theorem [47], and claimed in [30, Corollary 4.25] a kernel with at most $2k$ vertices on general graphs using spanning trees. Unfortunately, this latter kernelization algorithm is incorrect, as we discuss at the end of Section 3.

Boria et al. [16] initiated a study of the complexity of MMVC and presented a number of results, in particular a polynomial-time approximation algorithm with ratio $n^{1/2}$ on n-vertex graphs, and showed that, unless P = NP, no polynomial-time approximation algorithm with ratio $n^{1/2-\varepsilon}$ exists for any $\varepsilon > 0$. They also presented FPT algorithms for MMVC for several choices of the parameters such as the treewidth, the size of a maximum matching, or the size of a minimum vertex cover of the input graph. The authors asked explicitly whether kernels of size $o(k^2)$ exist for MMVC parameterized by k.

Zehavi [50] presented tight FPT algorithms, under the Strong Exponential Time Hypothesis, for MMVC and its weighted version parameterized by the size of a minimum vertex cover. Recently, Bonnet and Paschos [14] and Bonnet et al. [13] considered the inapproximability of MMVC in subexponential time.

Note that the MMVC problem is the dual of the well-studied Minimum Independent Dominating Set problem (to see this, note that the complement of any minimal vertex cover is an independent dominating set), which has applications in wireless and ad-hoc networks [42]. We refer to the survey of Goddard and Henning [35].

Our results and techniques. In this article we focus on the kernelization of the MMVC problem, which has been almost unexplored so far in the literature. Motivated by the question of Boria et al. [16] about the existence of subquadratic kernels for MMVC, we introduce a generic framework to obtain lower bounds on the degree of a "certain type" of polynomial kernels for parameterized vertex-maximization problems (in particular, for MMVC), based on a hypothesis that guarantees an inapproximability result, typically $\mathsf{P} \neq \mathsf{NP}$. Informally, by "certain type" we mean kernelization algorithms that, in polynomial time, either decide the instance (by answering "yes" or "no") or produce an equivalent instance of the considered problem in which the value of an optimal solution is "preserved", in the sense that it may drop only by the drop suffered by the parameter; see Definition 6 for the formal details. We call such kernels *large optimal preserving kernels*, or lop-*kernels* for short. Even if this type of kernels may seem restrictive, we are not aware of any known polynomial kernel for a vertex-maximization problem, such as those that have become nowadays standard [31], which is *not* a lop-kernel. The idea of our approach is to show (Theorem 3) that a lop-kernel yields a polynomial-time approximation algorithm whose ratio depends on the degree of the kernel, and to use known inapproximability results to obtain the desired lower bound.

Combining Theorem 3 (for $r = \frac{1}{2}$) with the known $\mathcal{O}(n^{\frac{1}{2}-\varepsilon})$-inapproximability result for MMVC by Boria et al. [16] immediately rules out the existence of a lop-kernel for MMVC with $\mathcal{O}(k^{2-\varepsilon})$ vertices for any $\varepsilon > 0$, unless $\mathsf{P} = \mathsf{NP}$. Thus, while Corollary 4 does not completely rule out the existence of subquadratic kernels for MMVC, it tells that, if such a kernel exists, it should consist of "non-standard" reduction rules.

Interestingly, our framework has consequences beyond the MMVC problem, namely for the Maximum Minimal Feedback Vertex Set (MMFVS) problem, defined in the natural way. Dublois et al. [27] recently provided a cubic kernel for MMFVS parameterized by the solution size, and proved that the problem does not admit an $\mathcal{O}(n^{\frac{2}{3}-\varepsilon})$-approximation algorithm for any $\varepsilon > 0$, unless $\mathsf{P} = \mathsf{NP}$. Hence, by applying Theorem 3 with $r = \frac{2}{3}$ we obtain (Corollary 5) that the cubic kernel of Dublois et al. [27] is essentially optimal.

In Section 4 we translate our framework to vertex-minimization problems, whose applicability is summarized in Theorem 13. Compared to existing frameworks to obtain lower bounds on kernelization, such as cross-compositions [8, 10], weak compositions [23, 24, 40], polynomial parameter transformations [6, 11], or techniques to obtain lower bounds on the coefficients of linear kernels [18], or that relate approximation and kernelization [1, 7, 37, 43, 45], our approach has the advantages that it is quite simple, straightforward to apply, and on same hypothesis on which the inapproximability result is based, typically $\mathsf{P} \neq \mathsf{NP}$. On the negative side, it has the following two drawbacks. The first one is that it can only be applied to vertex-maximization (or minimization) problems which are very hard to approximate, namely within a factor $\mathcal{O}(n^{r-\varepsilon})$ for some constant $r > 0$. Finally, our techniques are able to rule out the existence of what we call lop-kernels of certain sizes, but smaller non-standard kernels that do not preserve the value of large optimal solutions might, a priori, still exist. Hence, since our framework seems to be orthogonal to existing ones, we think that it adds to the above list of techniques to obtain kernelization lower bounds.

Coming back to the MMVC problem parameterized by the solution size, given the above negative result on general graphs, we identify graph classes where MMVC is still NP-hard and admits a subquadratic kernel. In particular, we deal with graph classes defined by excluding an *induced* subgraph H that satisfies the *Erdős-Hajnal property* [29], that is, for which there exists a constant $\delta > 0$ such that every H-free graph on n vertices contains either a clique or an independent set of size n^{δ}. In particular, we present a kernel for MMVC with $\mathcal{O}(k^{7/4})$ vertices on the well-studied class of bull-free graphs (Theorem 15), with $\mathcal{O}(k^{\frac{2t-3}{t-1}})$

vertices on K_t-free graphs for every $t \geq 3$, and with $\mathcal{O}(k^{5/3})$ vertices on paw-free graphs. The latter two results can be found in the full version [3]. To the best of our knowledge, this is the first time that the Erdős-Hajnal property is used to obtain polynomial kernels (we would like to note that it was used by Kratsch et al. [44] to obtain kernelization lower bounds).

Our strategy to obtain these subquadratic kernels on H-free graphs is as follows. By the high-degree rule mentioned above, given an instance (G, k), we may assume that the maximum degree of G is at most $k-1$. We find greedily a minimal vertex cover X of G. If $|X| \geq k$ we are done, so we may assume that $|X| \leq k-1$, hence the goal is to bound the size of $S := V(G) \setminus X$. Using that $G[X]$ is also H-free, the Erdős-Hajnal property implies (Lemma 14) that X can be partitioned in polynomial time into a sublinear (in k) number of independent sets and cliques. Since S is an independent set and we may assume that G has no isolated vertices, in order to bound $|S|$ by a subquadratic function of k, it is enough to show that, for each of the sublinearly many cliques or independent sets Y that partition X, its neighborhood in S has size $\mathcal{O}(k)$. This is easy if Y is an independent set: if $|N_S(Y)| \geq k$ we can conclude that (G, k) is a yes-instance (Lemma 2), so we may assume that $|N_S(Y)| \leq k-1$. The case where Y is a clique is more interesting, and we need ad-hoc arguments depending on each particular excluded induced subgraph H. In the full version [3] we also present several positive results for MMVC restricted to other graph classes, such as $K_{1,t}$-free graphs, or graph classes with bounded cliquewidth or chromatic number.

Finally, we show (Theorem 17) that MMVC, parameterized by the size of a minimum vertex cover (or of a maximum matching) of the input graph, does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly, even restricted to bipartite graphs. This result complements the FPT algorithms for MMVC under these parameterizations given by Boria et al. [16] and Zehavi [50], and shows that, in what concerns the existence of polynomial kernels for MMVC, the most natural structural parameters smaller than the solution size are not large enough to yield polynomial kernels (note that the treewidth of any graph is at most one more than its vertex cover number, hence our result rules out the existence of polynomial kernels for MMVC parameterized by treewidth as well). The proof consists of a polynomial parameter transformation from Monotone Sat parameterized by the number of variables. In particular, our reduction yields also the NP-hardness of MMVC on bipartite graphs, which provides an alternative proof to the one of Boliac and Lozin [12] via the NP-hardness of Minimum Independent Dominating Set on bipartite graphs.

Organization. Due to space limitations, some of the contents have been moved to full version [3]. In Section 2 we provide some basic preliminaries about graphs, the MMVC problem, and parameterized complexity. In Section 3 we state our framework to obtain kernelization lower bounds and present its consequences for MMVC and MMFVS. We present in Section 4 our framework for vertex-minimization problems, but due to space constraints we only provide the basic definitions and the main result (Theorem 13). We discuss in Section 5 the flaw in the linear kernel for MMVC claimed by Fernau [30]. Section 6 is devoted to the subquadratic kernel on bull-free graphs, which captures the main ideas of our approach using the Erdős-Hajnal property. Further subquadratic kernels and other positive results for MMVC can be found in the full version [3]. Our reduction to rule out the existence of polynomial kernels for MMVC parameterized by the size of a minimum vertex cover is presented in Section 7. We conclude the article in Section 8 with a discussion and some directions for further research.

2 Preliminaries

Graphs and functions. We use standard graph-theoretic notation, and we refer the reader to [25] for any undefined notation. For an integer $p \geq 1$, we let $[p]$ be the set containing all integers i with $1 \leq i \leq p$. We use $\uplus$ to denote the disjoint union. We will only consider finite undirected graphs without loops nor multiple edges, and we denote an edge between two vertices u and v by $\{u, v\}$. A subgraph H of a graph G is *induced* if H can be obtained from G by deleting a set of vertices $D = V(G) \setminus S$, and we denote $H = G[S]$. A graph G is *H-free* if it does not contain any induced subgraph isomorphic to H. If $\mathcal{H}$ is a collection of graphs, a graph G is *$\mathcal{H}$-free* is it is H-free for every $H \in \mathcal{H}$. For a graph G and a set $S \subseteq V(G)$, we use the notation $G \setminus S = G[V(G) \setminus S]$, and for a vertex $v \in V(G)$, we abbreviate $G \setminus \{v\}$ as $G \setminus v$. A vertex v is *complete* to a set $S \subseteq V(G)$ if v is adjacent to every vertex in S.

The *open* (resp. *closed*) *neighborhood* of a vertex v is denoted by $N(v)$ (resp. $N[v]$), whenever the graph G is clear from the context. For vertex sets $X, Y \subseteq V(G)$, we define $N[X] = \bigcup_{v \in X} N[v]$, $N(X) = N[X] \setminus X$, $N_Y[X] = N[X] \cap Y$, and $N_Y(X) = N_Y[X] \setminus X$. The *degree* of a vertex v in a graph G is defined as $|N(v)|$, and we denote it by $\mathsf{deg}_G(v)$, or just $\mathsf{deg}(v)$ of the graph is clear from the context. For an integer $t \geq 1$, we denote by P_t (resp. I_t, K_t) the path (resp. edgeless graph, complete graph) on t vertices. For two integers $a, b \geq 1$, we denote by $K_{a,b}$ the bipartite graph with parts of sizes a and b.

A *clique* (resp. *independent set*) of a graph G is a set of vertices that are pairwise adjacent (resp. not adjacent). A graph property is *hereditary* if whenever it holds for a graph G, it holds for all its induced subgraphs as well. Note that the properties of being an edgeless or a complete graph or an independent set are hereditary. We denote by $\Delta(G)$ (resp. $\omega(G)$ the maximum vertex degree (resp. clique size) of a graph G.

A *vertex cover* of a graph G is a set of vertices containing at least one endpoint of every edge, and it is *minimal* if no proper subset of it is a vertex cover. The main problem we study in the paper is formally stated as follows. We state it as a decision problem, since most of our results consider its parameterization by the solution size k.

Maximum Minimal Vertex Cover (MMVC)
Input: A graph G and a positive integer k.
Question: Does G contain a minimal vertex cover of size at least k?

The following observation has been already used in previous work [16, 50].

▶ **Observation 1.** *Let G be a graph. A set $X \subseteq V(G)$ is a minimal vertex cover of G if and only if X is a vertex cover of G and, for every vertex $v \in X$, $N(v) \nsubseteq X$.*

The next lemma provides a useful way to conclude that we are dealing with a yes-instance in our kernelization algorithms.

▶ **Lemma 2.** *Let G be a graph and let $S \subseteq V(G)$ be an independent set. There exists a minimal vertex cover of G containing $N(S)$.*

Proof. Note that, since S is an independent set, $V(G) \setminus S$ is a vertex cover of G. Hence, there exists a minimal vertex cover X of G such that $X \subseteq V(G) \setminus S$. We claim that $N(S) \subseteq X$. Suppose for the sake of contradiction that there exists a vertex $v \in N(S)$ such that $v \notin X$. Since v has a neighbor u in S and $S \cap X = \emptyset$, the edge $\{u, v\}$ would not be covered by X. ◀

Note that, in particular, Lemma 2 implies that if (G, k) is an instance of the Maximum Minimal Vertex Cover problem and $v \in V(G)$ is a vertex of degree at least k, then we can conclude that (G, k) is a yes-instance. This will allow us to assume, in our kernelization algorithms, that $\Delta(G) \leq k - 1$.

Parameterized complexity. We refer the reader to [21, 26] for basic background on parameterized complexity, and we recall here only some basic definitions used in this article. A *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}$. 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 $f(k) \cdot |I|^c$. For instance, the Vertex Cover problem parameterized by the size of the solution is FPT.

For an instance (x, k) of a parameterized problem Q, a *kernelization algorithm* is an algorithm $\mathcal{A}$ that, in polynomial time, generates from (x, k) an equivalent instance (x', k') of Q such that $|x'| + k' \leq f(k)$, for some computable function $f : \mathbb{N} \to \mathbb{N}$, where $|x'|$ denotes the size of x'. If $f(k)$ is bounded from above by a polynomial of the parameter, we say that Q admits a *polynomial kernel*. In particular, if $f(k)$ is bounded by a linear (resp. quadratic) function, then we say that Q admits a *linear* (resp. *quadratic*) kernel.

A *polynomial parameter transformation*, abbreviated as PPT, is an algorithm that, given an instance (x, k) of a parameterized problem A, runs in time polynomial in $|x|$ and outputs an instance (x', k') of a parameterized problem B such that k' is bounded from above by a polynomial on k and (x, k) is positive if and only if (x', k') is positive. If a parameterized problem A does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly and there exists a PPT from A to a parameterized problem B, then B does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly either [21].

3 A general framework for ruling out certain polynomial kernels

In this section we introduce our generic framework to obtain lower bounds on the degree of a "certain type" of polynomial kernels, which we call lop-*kernels* (see Definition 8), for parameterized vertex-maximization problems. Namely, we prove the following theorem. Note that, when applying it to a concrete problem Π, the inapproximability of Π will rely on some complexity assumption, typically P $\neq$ NP.

▶ **Theorem 3.** *Let Π be a vertex-maximization problem whose decision version is in* NP*, and suppose that Π does not admit a polynomial-time approximation algorithm with ratio $\mathcal{O}(n^{r-\varepsilon})$ on n-vertex graphs for $r, \varepsilon \in (0, 1]$. Then Π parameterized by the solution size does not admit a* lop*-kernel with $\mathcal{O}(k^{\frac{1}{1-r}-\varepsilon'})$ vertices for $\varepsilon' = \frac{\varepsilon}{(1-r+\varepsilon)(1-r)}$ when $r \in (0, 1)$, or with $\mathcal{O}(k^{\frac{1}{\varepsilon}})$ vertices when $r = 1$.*

Boria et al. [16] proved that the Maximum Minimal Vertex Cover problem does not admit an $\mathcal{O}(n^{\frac{1}{2}-\varepsilon})$-approximation algorithm for any $\varepsilon > 0$, unless P = NP. Hence, by applying Theorem 3 with $r = \frac{1}{2}$ we obtain the following corollary.

▶ **Corollary 4.** Maximum Minimal Vertex Cover *parameterized by the solution size does not admit a* lop*-kernel with $\mathcal{O}(k^{2-\varepsilon})$ vertices for any $\varepsilon > 0$, unless* P = NP*.*

Dublois et al. [27] recently provided a cubic kernel for the Maximum Minimal Feedback Vertex Set problem (defined naturally) parameterized by the solution size, and proved that the problem does not admit an $\mathcal{O}(n^{\frac{2}{3}-\varepsilon})$-approximation algorithm for any $\varepsilon > 0$, unless P = NP. Hence, by applying Theorem 3 with $r = \frac{2}{3}$ we obtain the following corollary, which states that the cubic kernel of Dublois et al. [27] is essentially optimal.

▶ **Corollary 5.** Maximum Minimal Feedback Vertex Set *parameterized by the solution size does not admit a* lop*-kernel with $\mathcal{O}(k^{3-\varepsilon})$ vertices for any $\varepsilon > 0$, unless* P = NP*.*

In the remainder of this section we prove Theorem 3. In order to do so, we present simple self-contained ad-hoc arguments relating the kernel size to the approximability of the considered problem.

We define lop-rules for an arbitrary vertex-maximization problem Π such that the input is a graph G, the output is a subset $S \subseteq V(G)$ satisfying some conditions, and the goal is to maximize $|S|$. Given a graph G and an integer k, we say that (G, k) is a yes-*instance* of Π if $\mathsf{opt}_\Pi(G) \geq k$, where $\mathsf{opt}_\Pi(G)$ denotes the maximum size of a solution of Π in G.

▶ **Definition 6.** *A* large optimal preserving *reduction rule, or* lop-rule *for short, for a vertex-maximization problem Π, is a polynomial-time algorithm R that, given a pair (G, k), where G is a graph and k is a positive integer, computes another pair (G', k') with $0 \leq k' \leq k$ such that*

1. *if (G, k) is a* no*-instance of Π, then (G', k') is a* no*-instance of Π, and*
2. *if (G, k) is a* yes*-instance of Π, then $\mathsf{opt}_\Pi(G') \geq \mathsf{opt}_\Pi(G) - (k - k')$, implying that (G', k') is a* yes*-instance of Π.*

Note that Property 2 in Definition 6 is stronger than the implication "if (G, k) is a yes-instance of Π, then (G', k') is a yes-instance of Π", which would yield a classical kernelization algorithm. Indeed, when we consider how this latter implication is generally proved in safeness proofs of classical kernels, one of the following scenarios often occur:

(a) For every solution S in G there exists a solution S' in G' with $|S'| \geq |S| - (k - k')$.
(b) If there exists a solution S in G with $|S| \geq k$, then there exists a solution S' in G' with $|S'| \geq |S| - (k - k')$.
(c) If there exists a solution S in G with $|S| \geq k$, then there exists a solution S' in G' with $|S'| \geq k'$.

In Case (a), the rule preserves all optimal solutions, and it implies that $\mathsf{opt}_\Pi(G') \geq \mathsf{opt}_\Pi(G) - (k - k')$. In Case (b), the rule preserves only large optimal solutions, and it implies that if $\mathsf{opt}_\Pi(G) \geq k$, then $\mathsf{opt}_\Pi(G') \geq \mathsf{opt}_\Pi(G) - (k - k')$, implying Property 2 above; note that if $\mathsf{opt}_\Pi(G) < k$, then $\mathsf{opt}_\Pi(G')$ and $\mathsf{opt}_\Pi(G)$ are not necessarily related. Case (c) corresponds to the weaker and classical implication "if (G, k) is a yes-instance of Π, then (G', k') is a yes-instance of Π".

The following observation is an immediate consequence of the definition of a lop-rule.

▶ **Observation 7.** lop*-rules can be composed. Formally, consider two* lop*-rules R_1 and R_2. Then, the rule R that, given a instance (G, k), returns $R_2(R_1(G, k))$ is a* lop*-rule.*

A typical example of a lop-rule is when we can identify a "dominant" set of vertices that can be safely included into a solution. More precisely, consider a rule that finds a subset $T \subseteq V(G)$ and a graph G' such that there exists an optimal solution $S^\star$ in G such that $S^\star = T \cup S'$, where S' is a solution in G', and for every solution S' in G', $S' \cup T$ is a solution in G. Such a rule is a lop-rule, as we even fall into Case (a) described above.

Even if we are not aware of known reduction rules for vertex-maximization problems that are *not* lop-rules, we can artificially devise such an example. For instance, for the MMVC problem, given an instance (G, k), if there is a vertex that has more than k neighbors of degree one, we can safely delete all but any k of them to obtain a reduced graph G', and leave k unchanged. Note that this rule falls into Case (c) above, since by Lemma 2 both G and G' are yes-instances of MMVC, but it does not satisfy Property 2 in Definition 6, since $\mathsf{mmvc}(G)$ may be arbitrarily larger than $\mathsf{mmvc}(G')$.

If we defined a lop-kernel as an algorithm consisting only of lop-rules, we would exclude from being a lop kernel, for instance, a rule that detects a yes-instance as in the above paragraph. This justifies the next definition, where we allow lop-kernels to decide instances.

▶ **Definition 8.** *Let Π be a vertex-maximization problem and let $s : \mathbb{N} \to \mathbb{N}$ be a computable function. A* lop-kernel of size s *for Π parameterized by the solution size is a polynomial-time algorithm that takes as input an instance (G, k), produces a reduced instance (G', k') by applying a (possibly empty) sequence of* lop*-rules to (G, k), and either*

- *determines that (G', k') is a* yes*-instance or a* no*-instance, or*
- *outputs (G', k') with $|V(G')| \leq s(k)$.*

Note that, as lop-rules provide equivalent instances, if the lop-kernel falls into the first case (where it correctly decides (G', k')), then it also correctly decides (G, k). On the other hand, if it falls into the second case (where it outputs (G', k')), and the kernel has size $\mathcal{O}(k^c)$ for some constant $c \geq 1$, then Property 2 implies that $\mathsf{opt}_\Pi(G) \leq \mathsf{opt}_\Pi(G') + (k - k') \leq |V(G')| + k = \mathcal{O}(k^c)$. Hence, a lop-kernel of size $\mathcal{O}(k^c)$ yields a polynomial-time algorithm certifying either that $\mathsf{opt}_\Pi(G) \geq k$ (when it decides that (G, k) is a yes-instance), or that $\mathsf{opt}_\Pi(G) = \mathcal{O}(k^c)$ (when it decides that (G, k) is a no-instance, or outputs (G', k')).

Our next objective is to use such a polynomial-time algorithm in order to obtain an approximation algorithm for the considered problem. For this, we need the following definition, which is inspired by a similar notion introduced by Hochbaum and Shmoys [41], and referred to as *f-relaxed decision procedure* in [48]. Note that the definition in [41] is for a minimization problem, and their algorithm has either to certify that $\mathsf{opt}_\Pi(G) > k$, or to produce a solution (which is not required here) of size at most $f(k)$.

▶ **Definition 9.** *Let Π be a vertex-maximization problem and let $f : \mathbb{N} \to \mathbb{N}$ be a function. An* f-dual-approximation algorithm for Π *is a polynomial-time algorithm that, given a graph G and a positive integer k, concludes one of the following:*

- $\mathsf{opt}_\Pi(G) \geq k$.
- $\mathsf{opt}_\Pi(G) < f(k)$.

In the next lemma we prove that a lop-kernel of size s yields an f-dual-approximation algorithm (where f depends on s), which in turn yields a classical approximation algorithm whose ratio depends on s. For the latter implication, proved in Lemma 11, we restrict ourselves to functions s that are polynomial, since our goal is to obtain lower bounds on the degree of polynomial kernels.

▶ **Lemma 10.** *Let Π be a vertex-maximization problem and let $s : \mathbb{N} \to \mathbb{N}$ be a computable function. If Π parameterized by the solution size k admits a* lop*-kernel of size $s(k)$, then Π admits an f-dual-approximation algorithm with $f(k) := s(k) + k + 1$.*

Proof. Let A be a lop-kernel of size $s(k)$ for Π parameterized by the solution size. Given an instance (G, k), let (G', k') be the instance computed by A (using only lop-rules). Observe first that, according to Observation 7, we have that the sequence of lop-rules involved in the reduction from (G, k) to (G', k') is equivalent to a single lop-rule that transforms (G, k) into (G', k'). If A concludes that (G', k') is a yes-instance, then as a lop-rule provides an equivalent instance, we conclude that (G, k) is a yes-instance, and fall into the first item of Definition 9.

Otherwise, we claim that $\mathsf{opt}_\Pi(G) < f(k)$. If A concludes that (G', k') is a no-instance, then we conclude that (G, k) is a no-instance. This implies $\mathsf{opt}_\Pi(G) < k$, and we are done because $k \leq f(k)$ for every $k \geq 0$. It remains to consider the case where A outputs (G', k'). If $\mathsf{opt}_\Pi(G) < k$ then again we are done. Otherwise, as $\mathsf{opt}_\Pi(G) \geq k$, Property 2 in Definition 6 implies that $\mathsf{opt}_\Pi(G) \leq \mathsf{opt}_\Pi(G') + (k - k') \leq |V(G')| + k < s(k) + k + 1 = f(k)$. ◀

In the proof of the next lemma we need the hypothesis that decision version of the considered problem Π belongs to NP, to be able to verify in polynomial time if a vertex subset is a solution. Due to this, we also need this hypothesis in Theorem 3.

▶ **Lemma 11.** *Let Π be a vertex-maximization problem whose decision version is in* NP*, $c > 1$ be a real number, and $f : \mathbb{N} \to \mathbb{N}$ be a computable function with $f(k) = \mathcal{O}(k^c)$. If Π admits an f-dual-approximation algorithm, then Π admits a polynomial-time approximation algorithm with ratio $\mathcal{O}(n^{\frac{c-1}{c}})$ on n-vertex graphs.*

Proof. Let A be an f-dual-approximation algorithm for Π. We proceed to construct a polynomial-time approximation algorithm for Π with the claimed ratio[1]. Given an n-vertex graph G, let k_0 be the largest positive integer k such that algorithm A returns that $\mathsf{opt}_\Pi(G) \geq k$. Note that k_0 can be found in polynomial time by performing at most n calls to algorithm A. If $k_0 = 0$, we have that $\mathsf{opt}_\Pi(G) < f(0) = \mathcal{O}(1)$, and since the decision version of Π is in NP, we can find an optimal solution in polynomial time by verifying all vertex subsets of size at most $f(0)$. Otherwise, that is, when $k_0 \geq 1$, our approximation algorithm returns k_0. Let us prove that it provides the claimed approximation ratio.

We distinguish two cases depending on the value of k_0. Suppose first that $k_0 \geq n^{1/c}$. Since $\mathsf{opt}_\Pi(G) \leq n$, in this case we get that

$$\frac{\mathsf{opt}_\Pi(G)}{k_0} \leq \frac{n}{n^{1/c}} = n^{\frac{c-1}{c}}. \tag{1}$$

Otherwise, it holds that $k_0 < n^{1/c}$. By the definition of k_0 we have that $\mathsf{opt}_\Pi(G) < f(k_0+1) = \mathcal{O}((k_0+1)^c) = \mathcal{O}((k_0)^c)$. Thus, in this case we get that

$$\frac{\mathsf{opt}_\Pi(G)}{k_0} = \frac{\mathcal{O}((k_0)^c)}{k_0} = \mathcal{O}\left((k_0)^{c-1}\right) = \mathcal{O}\left(n^{\frac{c-1}{c}}\right). \tag{2}$$

Since in both cases we have a ratio of $\mathcal{O}(n^{\frac{c-1}{c}})$, the lemma follows. ◀

We finally have all the ingredients to prove Theorem 3.

Proof of Theorem 3. We prove that Π parameterized by the solution size does not admit a lop-kernel with $\mathcal{O}(k^{\frac{1}{1-r+\varepsilon}})$ vertices. Once this is proved, the theorem follows since $\frac{1}{1-r+\varepsilon}$ is equal to $\frac{1}{\varepsilon}$ when $r = 1$ and $\frac{1}{1-r} - \varepsilon'$ for $\varepsilon' = \frac{\varepsilon}{(1-r+\varepsilon)(1-r)}$ when $r \in (0,1)$.

Assume to the contrary that Π parameterized by the solution size k admits a lop-kernel with $\mathcal{O}(k^{\frac{1}{1-r+\varepsilon}})$ vertices. By Lemma 10 it follows that Π admits an f-dual-approximation algorithm with $f(k) = \mathcal{O}(k^{\frac{1}{1-r+\varepsilon}}) + k + 1 = \mathcal{O}(k^{\frac{1}{1-r+\varepsilon}})$. We use this to get a contradiction by applying Lemma 11 for $c = \frac{1}{1-r+\varepsilon}$ and obtain that Π admits a polynomial-time approximation algorithm with ratio $\mathcal{O}(n^{r-\varepsilon})$ on n-vertex graphs. ◀

4 Our framework for vertex-minimization problems

In this section we provide the definitions of lop-kernel for vertex-minimization problems and present the corresponding result, Theorem 13, which is the translation of Theorem 3 to vertex-minimization problems. All the details can be found in the full version [3].

[1] We consider here the problem of computing an approximation of the optimal value, and not constructing the corresponding solution. This is not restrictive, as the type of inapproximability results that allow the application of Theorem 3, such as [16, 27], also apply to this case.

We consider an arbitrary vertex-minimization problem Π such that the input is a graph G, the output is a subset $S \subseteq V(G)$ satisfying some conditions, and the goal is to minimize $|S|$. Given a graph G and an integer k, we say that (G, k) is a yes-*instance* of Π if $\mathsf{opt}_\Pi(G) \leq k$, where $\mathsf{opt}_\Pi(G)$ denotes the minimum size of a solution of Π in G. In order to state our general result, we first need to define lop-rules and lop-kernels for vertex-minimization problems.

▶ **Definition 12.** *A* large optimal preserving *reduction rule, or* lop-rule *for short, for a vertex-minimization problem* Π*, is a polynomial-time algorithm* R *that, given a pair* (G, k)*, where* G *is a graph and* k *is a positive integer, computes another pair* (G', k') *with* $0 \leq k' \leq k$ *such that*

1. *if* (G, k) *is a* yes-*instance of* Π*, then* (G', k') *is a* yes-*instance of* Π*, and*
2. *if* (G, k) *is a* no-*instance of* Π*, then* $\mathsf{opt}_\Pi(G') \geq \mathsf{opt}_\Pi(G) - (k - k')$*, implying that* (G', k') *is a* no-*instance of* Π*.*

Recall (cf. Definition 6) that a lop-rule for a vertex-maximization problem can be seen as a classical kernel whose property "if (G, k) is a yes-instance, then (G', k') is a yes-instance" is strengthened, whereas Property 2 in Definition 12 is a reinforcement of the classical implication "if (G, k) is a no-instance of Π, then (G', k') is a no-instance of Π". This apparent lack of symmetry is due to technical reasons that make it possible to prove Theorem 13. However, both versions look more similar if we rewrite them in the following way. Namely, the two properties of Definition 6 can be rewritten as

1. if $\mathsf{opt}_\Pi(G) < k$, then $\mathsf{opt}_\Pi(G') < k'$, and
2. if $\mathsf{opt}_\Pi(G) \geq k$, then $\mathsf{opt}_\Pi(G') \geq \mathsf{opt}_\Pi(G) - (k - k')$, implying that $\mathsf{opt}_\Pi(G') \geq k'$,

and the two properties of Definition 12 can be rewritten as

1. if $\mathsf{opt}_\Pi(G) \leq k$, then $\mathsf{opt}_\Pi(G') \leq k'$, and
2. if $\mathsf{opt}_\Pi(G) > k$, then $\mathsf{opt}_\Pi(G') \geq \mathsf{opt}_\Pi(G) - (k - k')$, implying that $\mathsf{opt}_\Pi(G') > k'$.

Indeed, observe that the above conditions are exactly the same in both definitions, up to strict inequalities. Note also that, in both cases, these rules preserve "large solutions" in G into G', and this is why we call both of them "lop-rules".

Once we have defined lop-rules for vertex-minimization problems, the definition of lop-*kernel for a vertex-minimization problem* is the same as for vertex-maximization problems, that is, the same as Definition 8 by just replacing "vertex-maximization" with "vertex-minimization".

Note that, as lop-rules provide equivalent instances, if the lop-kernel falls into the first case (where it correctly decides (G', k')), then it also correctly decides (G, k). On the other hand, if it falls into the second case (where it outputs (G', k')), and the kernel has size $\mathcal{O}(k^c)$ for some constant $c \geq 1$, then Property 2 implies that $\mathsf{opt}_\Pi(G) \leq \mathsf{opt}_\Pi(G') + (k - k') \leq |V(G')| + k = \mathcal{O}(k^c)$. Hence, a lop-kernel of size $\mathcal{O}(k^c)$ yields a polynomial-time algorithm certifying either that $\mathsf{opt}_\Pi(G) > k$ (when it decides that (G, k) is a no-instance), or that $\mathsf{opt}_\Pi(G) = \mathcal{O}(k^c)$ (when it either decides that (G, k) is a yes-instance, or outputs (G', k')).

As in the case of vertex-maximization problems, we can use such a polynomial-time algorithm in order to obtain an approximation algorithm for the considered problem, and this is the main idea behind the proof of Theorem 13, which can be found in the full version [3].

▶ **Theorem 13.** *Let* Π *be a vertex-minimization problem whose decision version in* NP*, and suppose that* Π *does not admit a polynomial-time approximation algorithm with ratio* $\mathcal{O}(n^{r-\varepsilon})$ *on* n*-vertex graphs for* $r, \varepsilon \in (0, 1]$*. Then* Π *parameterized by the solution size does not admit a* lop*-kernel with* $\mathcal{O}(k^{\frac{1}{1-r}-\varepsilon'})$ *vertices for* $\varepsilon' = \frac{\varepsilon}{(1-r+\varepsilon)(1-r)}$ *when* $r \in (0, 1)$*, or with* $\mathcal{O}(k^{\frac{1}{\varepsilon}})$ *vertices when* $r = 1$*.*

5 An attempt to obtain a linear kernel for MMVC

In this section we briefly explain the flaw in the linear kernel for MMVC claimed by Fernau [30, Corollary 4.25], and that is based on joint unpublished work with Dehne, Fellows, Prieto, and Rosamond. The kernelization algorithm is a small modification of a linear kernel for the Nonblocker Set problem presented by Ore [46]. A set of vertices S of a graph G is a *nonblocker* if its complement is a dominating set of G, that is, for every $u \in S$ there exists $v \notin S$ with $\{u, v\} \in E(G)$. In the Nonblocker Set problem, we are given a graph G and an integer parameter k, and the goal is to decide whether G contains a nonblocker of size at least k. Suppose for simplicity that G is connected. The idea is to consider an arbitrary spanning tree T of G, root it arbitrarily at a vertex r, and partition $V(G) = V_0 \uplus V_1$ such that the vertices in V_0 (resp. V_1) are within even (resp. odd) distance from r in T. By construction, each of V_0 and V_1 is a nonblocker in G, so if one of them has size at least k, we can answer "yes", and otherwise $|V(G)| \leq 2k$ and we are done.

Back to MMVC, it is observed in [30, Reduction rule 24] that a simple reduction rule allows to assume that no connected component of G is a clique (in particular, an isolated vertex). Assume again for simplicity that G is connected. It is then claimed in [30] that, using the same algorithm as for Nonblocker Set, the largest of V_0 and V_1, say V_0, can be always completed into a minimal vertex cover of G, which would immediately yield a kernel of size at most $2k$ for MMVC. Unfortunately, this claim is not true: when adding new vertices to V_0 in order to make it a vertex cover of G, we may lose the minimality property, and some vertices may need to be removed. For instance, let G be the graph obtained from a triangle on vertices u, v, w by adding $p \geq 2$ pendant vertices to each of u, v, and w. Let T be the spanning tree obtained from G by removing the edge $\{v, w\}$, and root T at vertex u. Then $|V_0| = 1 + 2p$ and $|V_1| = 2 + p$, so $|V_0| > |V_1|$, and note that the edge $\{v, w\}$ is the only edge of G not covered by V_0. But adding either of v or w to V_0, say v, results in a non-minimal vertex cover of G, and therefore the p pendant vertices adjacent to v have to be removed from V_0, which yields a set of size $2 + p < \frac{|V(G)|}{2} = \frac{3+3p}{2}$, where we have used that $p \geq 2$. In fact, deciding whether a set $S \subseteq V(G)$ can the extended to a minimal vertex cover of G is an NP-complete problem [17].

6 A subquadratic kernel for MMVC on bull-free graphs

In this section we present a subquadratic kernel for Maximum Minimal Vertex Cover restricted to bull-free graphs when the parameter is the solution size k. Subquadratic kernels on other graph classes, as well as other positive results, can be found in the full version [3], namely on K_t-free graphs, t-bull-free graphs (that generalize bull-free graphs), paw-free graphs, $K_{1,t}$-free graphs, graphs with bounded chromatic number, and graphs with bounded cliquewidth.

For a constant $\delta > 0$, a graph H is said to satisfy the *Erdős-Hajnal property with constant* δ if every H-free graph G on n vertices contains either a clique or an independent set of size n^δ. The (still open) Erdős-Hajnal conjecture [29] states that every graph H satisfies the Erdős-Hajnal property. As reported by Chudnovsky [19], the Erdős-Hajnal conjecture has been verified for only a small number of graphs, namely all graphs on at most four vertices, the *bull* (i.e., the graph obtained by adding a pendant vertex to two different vertices of a triangle), the complete graphs, and every graph that can be constructed from them using the so-called *substitution operation* [2], which we define later.

Since our goal is to use the Erdős-Hajnal property in order to obtain kernels for Maximum Minimal Vertex Cover, we need an algorithmic version of it. As defined by Bonnet et al. [15], for a constant $\delta > 0$, a graph H is said to satisfy the *constructive Erdős-Hajnal property with constant* δ if there exists an algorithm that takes as input an H-free graph G on n vertices, and outputs in polynomial-time a clique or an independent set of G of size at least n^δ. Fortunately for our purposes, all the graphs H shown to satisfy the Erdős-Hajnal property so far, also satisfy its constructive version [15].

In the following lemma, we show that, if H is a graph satisfying the constructive Erdős-Hajnal property, then the vertex set of an H-free graph can be partitioned in polynomial time into "few" cliques or independent sets. This partition will then be used to obtain subquadratic kernels on H-free graphs for several graphs H.

▶ **Lemma 14.** *Let H be a graph satisfying the constructive Erdős-Hajnal property with constant δ. The vertex set of any H-free graph G on n vertices can be partitioned in polynomial time into a collection of cliques $\mathcal{C}$ and a collection of independent sets $\mathcal{I}$ such that $|\mathcal{C}| + |\mathcal{I}| \leq \left(\frac{1}{2^{(1-\delta)}-1}\right) \cdot n^{1-\delta}$.*

Proof. Let G be an H-free graph on n vertices. We initialize $X_0 = V(G), \mathcal{C} = \mathcal{I} = \emptyset$, and we run the following procedure as far as $|X_0| \geq 1$:

> Find in polynomial time a clique or an independent set Y in $G[X_0]$ with $|Y| \geq |X_0|^\delta$. Note that this is possible since $G[X_0]$ is an H-free graph for any $X_0 \subseteq V(G)$. Add Y to $\mathcal{C}$ or to $\mathcal{I}$ depending on whether Y is a clique or an independent set, respectively (if $|Y| = 1$, choose $\mathcal{C}$ or $\mathcal{I}$ arbitrarily). Update $X_0 \leftarrow X_0 \setminus Y$.

Clearly, the above algorithm terminates in polynomial time. It remains to bound $|\mathcal{C}| + |\mathcal{I}|$, which is equal to the number of iterations of the algorithm. To this end, for a positive integer i, we say that an iteration belongs to *step* i of the algorithm if the current set X_0 at the start of the iteration satisfies $\frac{n}{2^i} < |X_0| \leq \frac{n}{2^{i-1}}$. We denote by t_i the number of iterations of the algorithm within step i. By definition, $|\mathcal{C}| + |\mathcal{I}| = \sum_{i=1}^{\infty} t_i$. Let Y be a clique or an independent set found by the algorithm within step i. Since the current set X_0 satisfies $|X_0| > \frac{n}{2^i}$, we have that $|Y| > \left(\frac{n}{2^i}\right)^\delta$. And since the sum of the sizes of the sets found before the last iteration of step i is at most $\frac{n}{2^i}$, it follows that $t_i \leq \left(\frac{n}{2^i}\right)^{1-\delta}$. Note that, in particular, $t_i = 0$ for $i > \lceil \log n \rceil$. Therefore, we conclude that

$$|\mathcal{C}| + |\mathcal{I}| \;=\; \sum_{i=1}^{\infty} t_i \;\leq\; \sum_{i=1}^{\infty} \left(\frac{n}{2^i}\right)^{1-\delta} \;=\; n^{1-\delta} \cdot \sum_{i=1}^{\infty} \left(\frac{1}{2^{1-\delta}}\right)^i \;=\; n^{1-\delta} \cdot \left(\frac{1}{2^{(1-\delta)}-1}\right),$$

and the lemma follows. ◀

We are now ready to present the subquadratic kernel on bull-free graphs. Note that, since bipartite graphs are bull-free, MMVC restricted to bull-free graphs is NP-hard by [12] (or by Theorem 17). In the kernels presented in this section and in the full version, since we can easily obtain explicit constants, we decided not to use the big-O notation.

▶ **Theorem 15.** Maximum Minimal Vertex Cover *parameterized by k restricted to bull-free graphs admits a kernel with at most $c(k-1)^{7/4} + k - 1$ vertices, where $c = \frac{2}{2^{\frac{3}{4}}-1} < 3$.*

Proof. Let (G, k) be an instance of the Maximum Minimal Vertex Cover problem, where G is a bull-free graph. Recall that by Lemma 2 we can assume that the maximum degree of G is at most $k-1$. We start by finding greedily, starting from $V(G)$, a minimal

vertex cover X of G. Note that X can be easily found in polynomial time by Observation 1. If $|X| \geq k$, we conclude that (G, k) is a yes-instance, so we can assume that $|X| \leq k-1$. Let $S = V(G) \setminus X$ and note that S is an independent set.

Since the bull satisfies the constructive Erdős-Hajnal property with constant $\delta = \frac{1}{4}$ [15,20], we can apply Lemma 14 to the bull-free graph $G[X]$ and obtain in polynomial time a partition of X into a collection of cliques $\mathcal{C}$ and a collection of independent sets $\mathcal{I}$ such that $|\mathcal{C}| + |\mathcal{I}| \leq d \cdot |X|^{3/4} \leq d \cdot (k-1)^{3/4}$, where $d = \frac{1}{2^{\frac{3}{4}}-1} < 1.47$. Since we can assume that G has no isolated vertices, as they can be safely removed without affecting the type of the instance, it follows that

$$S = \bigcup_{C \in \mathcal{C}} N_S(C) \cup \bigcup_{I \in \mathcal{I}} N_S(I). \tag{3}$$

Hence, our objective is to bound $|N_S(Y)|$ for every $Y \in \mathcal{C} \cup \mathcal{I}$. Suppose first that $I \in \mathcal{I}$ is an independent set. From Lemma 2, if $|N_S(I)| \geq k$ we can conclude that (G, k) is a yes-instance, so we can assume henceforth that

$$\text{for every independent set } I \in \mathcal{I}, \text{ it holds } \ |N_S(I)| \leq k-1. \tag{4}$$

Suppose now that $C \in \mathcal{C}$ is a clique. We partition $N_S(C) = S_C^1 \uplus S_C^2$ as follows. Let S_C^1 be an inclusion-wise maximal set of vertices in $N_S(C)$ such that for any two (not necessarily distinct) vertices $x, y \in S_C^1$, $|N_C(x) \cup N_C(y)| \leq |C| - 1$. That is, S_C^1 is a maximal set in $N_S(C)$ such that the neighborhoods of its vertices pairwise do *not* cover the whole clique C. We let $S_C^2 = N_S(C) \setminus S_C^1$. The following is the crucial property of the set S_C^1.

▷ Claim 16. The vertices in S_C^1 can be ordered $x_1, \ldots, x_p$ so that $N_C(x_i) \subseteq N_C(x_j)$ if $i \leq j$.

Proof. In order to prove the claim, it is sufficient to prove that, for any two vertices $x, y \in S_C^1$, either $N_C(x) \subseteq N_C(y)$ or $N_C(y) \subseteq N_C(x)$. Suppose for the sake of contradiction that there exist two vertices $u \in N_C(x) \setminus N_C(y)$ and $v \in N_C(y) \setminus N_C(x)$. By definition of the set S_C^1, there exists a vertex $w \in C \setminus (N_C(x) \cup N_C(y))$. But then the vertices x, y, u, v, w induce a bull as illustrated in Figure 1, contradicting the hypothesis that G is bull-free. ◁

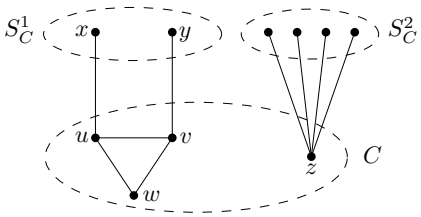


Figure 1 Configuration considered in the proof of Claim 16 and a vertex $z \in \bigcap_{x \in S_C^2} N_C(x)$.

Claim 16 implies in particular that, unless $S_C^1 = \emptyset$, there exists a vertex $u \in \bigcap_{x \in S_C^1} N_C(x)$. Since u has degree at most $k-1$ in G, and each vertex $x \in S_C^1$ is adjacent to u, it follows that $|S_C^1| \leq k-1$. Let us now focus on the set S_C^2. The definition of the set S_C^1 together with Claim 16 imply that there exists a vertex $z \in C \setminus \bigcup_{y \in S_C^1} N_C(y)$. Consider now an arbitrary vertex $x \in S_C^2$. Since x could not be added to S_C^1, there exists a vertex $y \in S_C^1$ such that $N_C(x) \cup N_C(y) = C$. But since $z \in C \setminus \bigcup_{y \in S_C^1} N_C(y)$, necessarily $z \in N_C(x)$. It follows that $z \in \bigcap_{x \in S_C^2} N_C(x)$ (see Figure 1). Using again the fact that z has degree at most $k-1$ in G, we obtain that $|S_C^2| \leq k-1$. Summarizing, we have that

$$\text{for every clique } C \in \mathcal{C}, \text{ it holds } \ |N_S(C)| = |S_C^1| + |S_C^2| \leq 2(k-1). \tag{5}$$

Putting all pieces together, Equations (3), (4), and (5) and the fact that $|X| \leq k-1$ and $|\mathcal{C}|+|\mathcal{I}| \leq d \cdot |X|^{3/4}$ imply that, unless we have already concluded that (G,k) is a yes-instance,

$$\begin{aligned}|V(G)| = |X| + |S| \;&=\; |X| + |\bigcup_{C\in\mathcal{C}} N_S(C)| + |\bigcup_{I\in\mathcal{I}} N_S(I)| \\ &\leq |X| + (|\mathcal{C}|+|\mathcal{I}|) \cdot \max_{Y\in\mathcal{C}\cup\mathcal{I}} |N_S(Y)| \;\leq\; k-1+d\cdot(k-1)^{3/4}\cdot 2(k-1) \\ &= 2d\cdot(k-1)^{7/4}+k-1, \quad \text{and the theorem follows.}\end{aligned}$$

◀

7 Ruling out polynomial kernels for MMVC for smaller parameters

In this section we rule out, assuming that NP $\not\subseteq$ coNP/poly, the existence of polynomial kernels for MMVC parameterized by the size of minimum vertex cover of the input graph. As mentioned in the introduction, the reduction given in Theorem 17 also provides an alternative proof of the NP-completeness of MMVC on bipartite graphs, which also follows from [12]. We note that the existing NP-hardness reductions for MMVC, such as the one in [12], do *not* seem to be easily modifiable so to yield the non-existence of polynomial kernels.

▶ **Theorem 17.** *The* Maximum Minimal Vertex Cover *problem parameterized by the size of a minimum vertex cover (or of a maximum matching) of the input graph does not admit a polynomial kernel unless* NP $\subseteq$ coNP/poly*, even restricted to bipartite graphs.*

Proof. We present a PPT from Monotone Sat parameterized by the number of variables, which is also an NP-completeness reduction. The Monotone Sat problem is the restriction of the Sat problem to formulas in which the literals in each clause are either all positive or all negative. This problem is well-known to be NP-complete [33], and it is easy to see that, when parameterized by the number of variables, it does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly. Indeed, Fortnow and Santhanam [32] proved that the Sat problem parameterized by the number of variables does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly, and the classical reduction from Sat to Monotone Sat that replaces each variable with a "positive" and a "negative" variable and adds extra clauses appropriately [33] is in fact a PPT when the parameter is the number of variables.

Given an instance ϕ of Monotone Sat, where the formula ϕ contains n variables and m clauses, we construct in polynomial time an instance (G,k) of Maximum Minimal Vertex Cover as follows. For each variable x_i of ϕ, $i \in [n]$, we add to G four vertices $\ell_i, x_i^+, x_i^-, r_i$ and three edges $\{\ell_i, x_i^+\}, \{x_i^+, x_i^-\}, \{x_i^-, r_i\}$, hence inducing a P_4. We call the vertex x_i^+ (resp. x_i^-) a *positive* (resp. a *negative*) vertex of G. For each clause C_j of ϕ, $j \in [m]$, we add to G a vertex c_j, which we connect to the positive or negative vertices corresponding to the literals contained in C_j. This concludes the construction of G, which is illustrated in Figure 2(a). Note that, since ϕ is a monotone formula, G is a bipartite graph. Note also that the set of vertices $\{x_i^+, x_i^- \mid i \in [n]\}$ is a minimum vertex cover of G of size $2n$, and that the set of edges $\{\{\ell_i, x_i^+\}, \{x_i^-, r_i\} \mid i \in [n]\}$ is a maximum matching of G of size $2n$. We claim that ϕ is satisfiable if and only if G contains a minimal vertex cover of size $k := 2n+m$.

Suppose first that ϕ is satisfiable, and let σ be an assignment of the variables that satisfies all the clauses in ϕ. We proceed to define a minimal vertex cover X of G of size k. First, add to X all the clause vertices $\{c_j \mid j \in [m]\}$. For every $i \in [n]$, if $\sigma(x_i) =$ true (resp. $\sigma(x_i) =$ false), add to X vertices x_i^- and ℓ_i (resp. x_i^+ and r_i). See Figure 2(b) for an illustration, where the set X is shown with larger red vertices. Clearly, X is a vertex cover of G. To see that it is minimal, by Observation 1 it is enough to verify that, for every vertex $v \in X$, $N[v] \not\subseteq X$. This condition holds easily for all vertices in X that are in the P_4's,

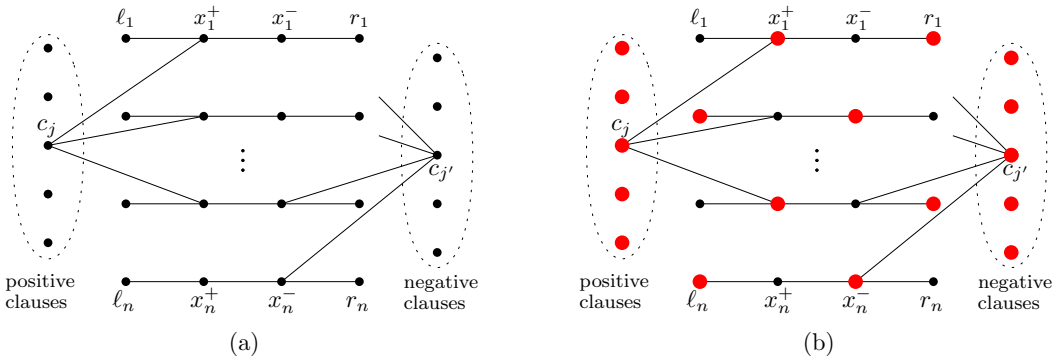


Figure 2 (a) Illustration of the graph G built from the formula ϕ in the proof of Theorem 17. (b) A minimal vertex cover X of G is shown with larger red vertices.

since for each P_4 its vertices in X are not adjacent. Let c_j be a clause vertex. Since σ is a satisfying assignment of the variables, there exists a variable x_i such that if $\sigma(x_i) = \mathsf{true}$ (resp. $\sigma(x_i) = \mathsf{false}$) then $x_i \in C_j$ (resp. $\bar{x}_i \in C_j$). By definition of X, if $\sigma(x_i) = \mathsf{true}$ (resp. $\sigma(x_i) = \mathsf{false}$) then $x_i^+ \notin X$ (resp. $x_i^- \notin X$), and by construction of G we have that $x_i^+ \in N(c_j)$ (resp. $x_i^- \in N(c_j)$), so in both cases $N[c_j] \not\subseteq X$.

Conversely, suppose that G contains a minimal vertex cover X of size k, and we proceed to define a variable assignment σ as follows. For $i \in [n]$, as $\{x_i^+, x_i^-\} \in E(G)$ we have that X contains one or two vertices in the set $\{x_i^+, x_i^-\}$. If $x_i^+ \notin X$ (resp. $x_i^- \notin X$) we set $\sigma(x_i) = \mathsf{true}$ (resp. $\sigma(x_i) = \mathsf{false}$), and if both x_i^+ and x_i^- belong to X we set $\sigma(x_i)$ to true or to false arbitrarily. We claim that σ satisfies all the clauses in ϕ. For $i \in [n]$, let P^i be the P_4 of G induced by the vertices $\ell_i, x_i^+, x_i^-, r_i$. Since X is a vertex cover, clearly $|X \cap V(P^i)| \geq 2$. We claim that $|X \cap V(P^i)| = 2$. Indeed, if $|X \cap V(P^i)| \geq 3$, then $\{\ell_i, x_i^+\} \subseteq X$ or $\{x_i^-, r_i\} \subseteq X$ (or both). But then $N[\ell_i] \subseteq X$ or $N[r_i] \subseteq X$ (or both), contradicting Observation 1. Thus, $|X \cap V(P^i)| = 2$, which implies that $|X \cap \bigcup_{i \in [n]} V(P^i)| = 2n$, hence necessarily X contains the whole set $\{c_j \mid j \in [m]\}$ of clause vertices. Consider an arbitrary clause vertex c_j. Since X is minimal and $c_j \in X$, by Observation 1 there exists a neighbor of c_j in G that is *not* in X, and by definition of σ it follows that the literal corresponding to that neighbor of c_j satisfies clause C_j. Thus, σ is a satisfying assignment and the proof is complete.

Finally, note that the above reduction is also an NP-completeness reduction from Monotone Sat to Maximum Minimal Vertex Cover on bipartite graphs. ◀

8 Conclusions and further research

We presented a framework to obtain lower bounds on the degrees of certain types of polynomial kernels, which we called lop-kernels, for vertex-maximization and vertex-minimization problems. Note that the classical kernels for Vertex Cover such as those using the high-degree rule, the crown decomposition rule, or the Nemhauser-Trotter rule [31], are lop-kernels. More involved kernels, such as those based on protrusion replacement [9], are also lop-kernels. Hence, the most natural question is whether the "lop" assumption could be dropped from our general results, namely Theorem 3 and Theorem 13. For the vertex-minimization version (Theorem 13), we know that this is not possible: the problem of deleting at most k vertices from an n-vertex graph in order to obtain a tree admits a kernel with $\mathcal{O}(k^4)$ vertices [34], but no $\mathcal{O}(n^{1-\varepsilon})$-approximation for any $\varepsilon > 0$ unless $\mathsf{P} \neq \mathsf{NP}$ [49]. Therefore, if a polynomial lop-kernel for this problem existed, it would contradict Theorem 13, assuming that $\mathsf{P} \neq \mathsf{NP}$.

Thus, the algebraic reduction rule presented by Giannopoulou et al. [34], which is based on identifying a subset of linear equations of appropriate size that captures all solutions of size at most k, cannot be (even transformed to) a lop-rule. We still do not know of a similar example that is a vertex-maximization problem.

We showed that a direct application of Theorem 3 yields kernelization lower bounds for MMVC (Corollary 4) and MMFVS (Corollary 5), matching the sizes of the best known kernels for these problems. We believe that our result could be applied to other vertex-maximization problems, in particular to the "max-min" version of other vertex-minimization problems, as they seem to be quite hard to approximate. It would be interesting to find examples of vertex-minimization problems where Theorem 13 could be applied. Here, the natural candidates seem to be the "min-max" version of vertex-maximization problems, which seem to have been almost unexplored so far. We are currently working on the adaptation of our framework to edge-maximization (or minimization) problems, and even to problems with more general objective functions.

We presented (Section 6) subquadratic kernels on H-free graphs for some graphs H satisfying the (constructive) Erdős-Hajnal property, such as the bull, the complete graphs, or the paw. It would be interesting to obtain subquadratic kernels for other graphs H satisfying the Erdős-Hajnal property, such as C_4, the diamond, P_5, or C_5. Note that, from [38], C_4 and the diamond satisfy the constructive Erdős-Hajnal property with constant $\delta \geq 1/3$. Note also that the graphs constructed in the reduction of Theorem 17 are $\{C_5, \text{diamond}\}$-free, as they are bipartite, hence MMVC is NP-hard on this class, in contrast to the fact (see the full version [3]) that MMVC can be solved in linear time on $\{P_5, \text{diamond}\}$-free graphs. To the best of our knowledge, the complexity on P_5-free graphs is open, as well as on $K_{1,t}$ graphs for $t \geq 3$ (see the full version [3]). It is worth mentioning that P_5-free graphs have unbounded cliquewidth, because co-bipartite graphs, which are P_5-free, have unbounded cliquewidth.

As defined in Section 3, for a graph G we denoted by $\mathsf{mmvc}(G)$ the maximum size of a minimal vertex cover of G. Boria et al. [16] proved that if G is an n-vertex graph without isolated vertices, then $\mathsf{mmvc}(G) \geq \lfloor n^{1/2} \rfloor$. Note that this immediately yields a quadratic kernel for MMVC: if $k \leq \lfloor n^{1/2} \rfloor$ we answer "yes", otherwise $n \leq k^2$. By the same argument, if $\mathcal{C}$ is a graph class such that every n-vertex graph $G \in \mathcal{C}$ without isolated vertices satisfies $\mathsf{mmvc}(G) \geq n^{1/2+\varepsilon}$, for some $\varepsilon > 0$, then MMVC restricted to $\mathcal{C}$ admits a (subquadratic) kernel with at most $k^{\frac{2}{1+2\varepsilon}}$ vertices. It might be possible that this is the case for some of the H-free graph classes for which we provided subquadratic kernels in Section 6: we were not able to find any counterexample, that is, a family of n-vertex H-free graphs G for which $\mathsf{mmvc}(G) = \Theta(n^{1/2})$. In particular, the case of triangle-free graphs seems particularly interesting. Haviland [39] and Goddard and Lyle [36] established upper bounds on the size of a minimum independent dominating set (that is, the complement of a minimal vertex cover) of triangle-free graphs. It follows from their results [36,39] that there exist n-vertex triangle-free graphs G with $\mathsf{mmvc}(G) = \Theta(n^{2/3} \cdot \log n)$, hence if such a constant $\varepsilon > 0$ as discussed above exists for triangle-free graphs, necessarily $\varepsilon \leq \frac{2}{3} - \frac{1}{2} = \frac{1}{6}$. Therefore, the smallest kernel that we may obtain in this way on triangle-free graphs would have $k^{\frac{2}{1+2\varepsilon}} \leq k^{3/2}$ vertices, which matches the size of the kernel that we obtained in the full version [3] for the particular case $t = 3$, disregarding lower-order terms and multiplicative constants. Finding such a constant $\varepsilon > 0$ on H-graphs for small graphs H, in particular on triangle-free graphs, looks like a challenging problem, having interesting connections with the Ramsey numbers [36, 39].

References

1. Faisal N. Abu-Khzam, Cristina Bazgan, Morgan Chopin, and Henning Fernau. Approximation algorithms inspired by kernelization methods. In *Proc. of the 25th International Symposium on Algorithms and Computation (ISAAC)*, volume 8889 of *LNCS*, pages 479–490, 2014. doi:10.1007/978-3-319-13075-0_38.
2. Noga Alon, János Pach, and József Solymosi. Ramsey-type theorems with forbidden subgraphs. *Combinatorica*, 21(2):155–170, 2001. doi:10.1007/s004930100016.
3. Júlio Araújo, Marin Bougeret, Victor A. Campos, and Ignasi Sau. A new framework for kernelization lower bounds: the case of Maximum Minimal Vertex Cover. Full version of the current article, 2021. arXiv:2102.02484.
4. Júlio Araújo, Marin Bougeret, Victor A. Campos, and Ignasi Sau. Parameterized complexity of computing maximum minimal blocking and hitting sets. Manuscript submitted for publication, 2021. arXiv:2102.03404.
5. Cristina Bazgan, Ljiljana Brankovic, Katrin Casel, Henning Fernau, Klaus Jansen, Kim-Manuel Klein, Michael Lampis, Mathieu Liedloff, Jérôme Monnot, and Vangelis Th. Paschos. The many facets of upper domination. *Theoretical Computer Science*, 717:2–25, 2018. doi:10.1016/j.tcs.2017.05.042.
6. Daniel Binkele-Raible, Henning Fernau, Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Yngve Villanger. Kernel(s) for problems with no kernel: On out-trees with many leaves. *ACM Transactions on Algorithms*, 8(4):38:1–38:19, 2012. doi:10.1145/2344422.2344428.
7. Arindam Biswas, Venkatesh Raman, and Saket Saurabh. Approximation in (poly-) logarithmic space. In *Proc. of the 45th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 170 of *LIPIcs*, pages 16:1–16:15, 2020. doi:10.4230/LIPIcs.MFCS.2020.16.
8. Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *Journal of Computer and System Sciences*, 75(8):423–434, 2009. doi:10.1016/j.jcss.2009.04.001.
9. Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (Meta) Kernelization. *Journal of the ACM*, 63(5):44:1–44:69, 2016.
10. 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.
11. Hans L. Bodlaender, Stéphan Thomassé, and Anders Yeo. Kernel bounds for disjoint cycles and disjoint paths. *Theoretical Computer Science*, 412(35):4570–4578, 2011. doi:10.1016/j.tcs.2011.04.039.
12. Rodica Boliac and Vadim V. Lozin. Independent domination in finitely defined classes of graphs. *Theoretical Computer Science*, 301(1-3):271–284, 2003. doi:10.1016/S0304-3975(02)00586-8.
13. Édouard Bonnet, Michael Lampis, and Vangelis Th. Paschos. Time-approximation trade-offs for inapproximable problems. *Journal of Computer and System Sciences*, 92:171–180, 2018. doi:10.1016/j.jcss.2017.09.009.
14. Édouard Bonnet and Vangelis Th. Paschos. Sparsification and subexponential approximation. *Acta Informatica*, 55(1):1–15, 2018. doi:10.1007/s00236-016-0281-2.
15. Édouard Bonnet, Stéphan Thomassé, Xuan Thang Tran, and Rémi Watrigant. An algorithmic weakening of the Erdős-Hajnal conjecture. In *Proc. of the 28th Annual European Symposium on Algorithms (ESA)*, volume 173 of *LIPIcs*, pages 23:1–23:18, 2020. doi:10.4230/LIPIcs.ESA.2020.23.
16. Nicolas Boria, Federico Della Croce, and Vangelis Th. Paschos. On the max min vertex cover problem. *Discrete Applied Mathematics*, 196:62–71, 2015. doi:10.1016/j.dam.2014.06.001.
17. Katrin Casel, Henning Fernau, Mehdi Khosravian Ghadikolaei, Jérôme Monnot, and Florian Sikora. Extension of Vertex Cover and Independent Set in Some Classes of Graphs. In *Proc. of the 11th International Conference on Algorithms and Complexity (CIAC)*, volume 11485 of *LNCS*, pages 124–136, 2019. doi:10.1007/978-3-030-17402-6_11.

18 Jianer Chen, Henning Fernau, Iyad A. Kanj, and Ge Xia. Parametric duality and kernelization: Lower bounds and upper bounds on kernel size. *SIAM Journal on Computing*, 37(4):1077–1106, 2007. doi:10.1137/050646354.

19 Maria Chudnovsky. The Erdős-Hajnal Conjecture - A Survey. *Journal of Graph Theory*, 75(2):178–190, 2014. doi:10.1002/jgt.21730.

20 Maria Chudnovsky and Shmuel Safra. The Erdős-Hajnal conjecture for bull-free graphs. *Journal of Combinatorial Theory, Series B*, 98(6):1301–1310, 2008. doi:10.1016/j.jctb.2008.02.005.

21 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.

22 Peter Damaschke. Parameterized algorithms for double hypergraph dualization with rank limitation and maximum minimal vertex cover. *Discrete Optimization*, 8(1):18–24, 2011. doi:10.1016/j.disopt.2010.02.006.

23 Holger Dell and Dániel Marx. Kernelization of packing problems. In *Proc. of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 68–81, 2012. doi:10.1137/1.9781611973099.6.

24 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.

25 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012. URL: https://dblp.org/rec/books/daglib/0030488.bib.

26 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.

27 Louis Dublois, Tesshu Hanaka, Mehdi Khosravian Ghadikolaei, Michael Lampis, and Nikolaos Melissinos. (In)approximability of Maximum Minimal FVS. In *Proc. of the 31st International Symposium on Algorithms and Computation (ISAAC)*, volume 181 of *LIPIcs*, pages 3:1–3:14, 2020. The cubic kernel appears in the full version, available at https://arxiv.org/abs/2009.09971. doi:10.4230/LIPIcs.ISAAC.2020.3.

28 Louis Dublois, Michael Lampis, and Vangelis Th. Paschos. Upper Dominating Set: Tight Algorithms for Pathwidth and Sub-Exponential Approximation. *CoRR*, abs/2101.07550, 2021. arXiv:2101.07550.

29 Paul Erdős and András Hajnal. Ramsey-type theorems. *Discrete Applied Mathematics*, 25(1-2):37–52, 1989. doi:10.1016/0166-218X(89)90045-0.

30 Henning Fernau. *Parameterized algorithms: a graph-theoretic approach*. Habilitationsschrift, Universität Tübingen, 2005. URL: http://www.informatik.uni-trier.de/~fernau/papers/habil.pdf.

31 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization. Theory of Parameterized Preprocessing*. Cambridge University Press, 2019. doi:10.1017/9781107415157.

32 Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *Journal of Computer and System Sciences*, 77(1):91–106, 2011. doi:10.1016/j.jcss.2010.06.007.

33 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. doi:https://dl.acm.org/doi/book/10.5555/574848.

34 Archontia C. Giannopoulou, Daniel Lokshtanov, Saket Saurabh, and Ondrej Suchý. Tree Deletion Set has a polynomial kernel but no $\mathrm{OPT}^{O(1)}$ approximation. *SIAM Journal on Discrete Mathematics*, 30(3):1371–1384, 2016. doi:10.1137/15M1038876.

35 Wayne Goddard and Michael A. Henning. Independent domination in graphs: A survey and recent results. *Discrete Mathematics*, 313(7):839–854, 2013. doi:10.1016/j.disc.2012.11.031.

36 Wayne Goddard and Jeremy Lyle. Independent dominating sets in triangle-free graphs. *Journal of Combinatorial Optimization*, 23(1):9–20, 2012. doi:10.1007/s10878-010-9336-4.

37 Jiong Guo, Iyad A. Kanj, and Stefan Kratsch. Safe approximation and its relation to kernelization. In *Proc. of the 6th International Symposium on Parameterized and Exact Computation (IPEC)*, volume 7112 of *LNCS*, pages 169–180, 2011. doi:10.1007/978-3-642-28050-4_14.

38 András Gyárfás. Reflections on a Problem of Erdős and Hajnal. In Ronald L. Graham, Jaroslav Nesetril, and Steve Butler, editors, *The Mathematics of Paul Erdős II*, pages 135–141. Springer, 2013. doi:10.1007/978-1-4614-7254-4_11.

39 Julie Haviland. Independent domination in triangle-free graphs. *Discrete Mathematics*, 308(16):3545–3550, 2008. doi:10.1016/j.disc.2007.07.010.

40 Danny Hermelin and Xi Wu. Weak compositions and their applications to polynomial lower bounds for kernelization. In *Proc. of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 104–113, 2012. doi:10.1137/1.9781611973099.9.

41 Dorit S. Hochbaum and David B. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM*, 34(1):144–162, 1987. doi:10.1145/7531.7535.

42 Johann L. Hurink and Tim Nieberg. Approximating minimum independent dominating sets in wireless networks. *Information Processing Letters*, 109(2):155–160, 2008. doi:10.1016/j.ipl.2008.09.021.

43 Stefan Kratsch. Polynomial kernelizations for MIN $F^+\Pi_1$ and MAX NP. *Algorithmica*, 63(1-2):532–550, 2012. doi:10.1007/s00453-011-9559-5.

44 Stefan Kratsch, Marcin Pilipczuk, Ashutosh Rai, and Venkatesh Raman. Kernel lower bounds using co-nondeterminism: Finding induced hereditary subgraphs. *ACM Transactions on Computation Theory*, 7(1):4:1–4:18, 2014. doi:10.1145/2691321.

45 Daniel Lokshtanov, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. Lossy kernelization. In *Proc. of the 49th Annual ACM Symposium on Theory of Computing (STOC)*, pages 224–237, 2017. doi:10.1145/3055399.3055456.

46 Oystein Ore. *Theory of Graphs*. American Mathematical Society Colloquium Publications, volume 38, 1962. URL: https://bookstore.ams.org/coll-38.

47 Neil Robertson, Daniel P. Sanders, Paul D. Seymour, and Robin Thomas. The four-colour theorem. *Journal of Combinatorial Theory, Series B*, 70(1):2–44, 1997. doi:10.1006/jctb.1997.1750.

48 David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011. URL: http://www.cambridge.org/de/knowledge/isbn/item5759340/?site_locale=de_DE.

49 Mihalis Yannakakis. The effect of a connectivity requirement on the complexity of maximum subgraph problems. *Journal of the ACM*, 26(4):618–630, 1979. doi:10.1145/322154.322157.

50 Meirav Zehavi. Maximum Minimal Vertex Cover Parameterized by Vertex Cover. *SIAM Journal on Discrete Mathematics*, 31(4):2440–2456, 2017. doi:10.1137/16M109017X.

CNF Satisfiability in a Subspace and Related Problems

Vikraman Arvind ✉
The Institute of Mathematical Sciences (HBNI), Chennai, India

Venkatesan Guruswami ✉
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

We introduce the problem of finding a satisfying assignment to a CNF formula that must further belong to a prescribed input subspace. Equivalent formulations of the problem include finding a point outside a union of subspaces (the Union-of-Subspace Avoidance (USA) problem), and finding a common zero of a system of polynomials over $\mathbb{F}_2$ each of which is a product of affine forms.

We focus on the case of k-CNF formulas (the k-Sub-Sat problem). Clearly, k-Sub-Sat is no easier than k-SAT, and might be harder. Indeed, via simple reductions we show that 2-Sub-Sat is NP-hard, and W[1]-hard when parameterized by the co-dimension of the subspace. We also prove that the optimization version Max-2-Sub-Sat is NP-hard to approximate better than the trivial 3/4 ratio even on satisfiable instances.

On the algorithmic front, we investigate fast exponential algorithms which give non-trivial savings over brute-force algorithms. We give a simple branching algorithm with running time $(1.5)^r$ for 2-Sub-Sat, where r is the subspace dimension, as well as an $O^*(1.4312)^n$ time algorithm where n is the number of variables.

Turning to k-Sub-Sat for $k \geqslant 3$, while known algorithms for solving a system of degree k polynomial equations already imply a solution with running time $\approx 2^{r(1-1/2k)}$, we explore a more combinatorial approach. Based on an analysis of critical variables (a key notion underlying the randomized k-SAT algorithm of Paturi, Pudlak, and Zane), we give an algorithm with running time $\approx \binom{n}{\leqslant t} 2^{n-n/k}$ where n is the number of variables and t is the co-dimension of the subspace. This improves upon the running time of the polynomial equations approach for small co-dimension. Our combinatorial approach also achieves polynomial space in contrast to the algebraic approach that uses exponential space. We also give a PPZ-style algorithm for k-Sub-Sat with running time $\approx 2^{n-n/2k}$. This algorithm is in fact oblivious to the structure of the subspace, and extends when the subspace-membership constraint is replaced by any constraint for which partial satisfying assignments can be efficiently completed to a full satisfying assignment. Finally, for systems of $O(n)$ polynomial equations in n variables over $\mathbb{F}_2$, we give a fast exponential algorithm when each polynomial has bounded degree irreducible factors (but can otherwise have large degree) using a degree reduction trick.

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

Keywords and phrases CNF Satisfiability, Exact exponential algorithms, Hardness results

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.5

Related Version *Full Version*: `https://arxiv.org/abs/2108.05914` [1]

Funding *Venkatesan Guruswami*: Portions of this work were done during visits to the Institute of Mathematical Sciences, Chennai. Research supported in part by the National Science Foundation grant CCF-1908125 and a Simons Investigator Award.

Acknowledgements We thank anonymous reviewers for useful comments and pointers to the literature.

1 Introduction

Given an n-variate Boolean formula Φ along with an affine subspace $A \subseteq \mathbb{F}_2^n$ (given by a system of $\mathbb{F}_2$-linear equations) as input, we explore the complexity of testing if Φ has a satisfying assignment in A. This is a natural twist on Boolean constraint satisfaction problems that studies the effects of linear algebra on Boolean logic. Our focus shall be on the case when Φ is presented in Conjunctive Normal Formal (CNF). We refer to this problem as *satisfiability in a subspace* and denote it by Sub-Sat. This framework can capture non-Boolean problems such as Graph K-Colorability indicating the richness of combining the problem of Boolean CNF-satisfiability with a linear-algebraic constraint. We also note that in the area of practical SAT solvers there is interest in CNF satisfiability conjuncted with XOR constraints [26, 25].

Further, Sub-Sat has two other equivalent interesting formulations. The first of these is *union of subspace avoidance*, USA for short: Given affine subspaces $A_1, A_2, \ldots, A_m \subseteq \mathbb{F}_2^n$ is there an $x \in \mathbb{F}_2^n$ that is not in the union $\bigcup_{i=1}^m A_i$? A different formulation is a special case of finding a solution to a bunch of polynomial equations $p_i = 0$ over $\mathbb{F}_2^n$, namely when each p_i is a product of affine forms. We refer to this reformulation as PAF-Sat. We will describe these (easy) equivalences in Section 1.3.

For most of the paper, we restrict attention to the case when Φ is a k-CNF formula (a CNF formula with clauses of width at most k) for a fixed k, referred to as the k-Sub-Sat problem. Clearly, k-Sub-Sat is a generalization of the well-studied k-Sat (k-CNF satisfiability). In terms of the two reformulations above, k-Sub-Sat corresponds to the USA problem when the spaces A_i have co-dimension at most k, and for the PAF-Sat problem, each polynomial p_i is the product of up to k affine forms.

We present both hardness results and algorithms for k-Sub-Sat, described in Sections 1.1 and 1.2 below respectively. Owing to the NP-hardness of the problems, the algorithmic focus is on exponential time algorithms that give non-trivial improvements over brute-force.

There are two possible angles from which to view the study of k-Sub-Sat. The first is as a problem intermediate between satisfiability of k-CNF formula and a system of degree k polynomial equations. The second is as a specific instance of a constraint satisfaction problem (CSP) obtained by combining two fundamental types of constraints. There have been a few works [20, 7] giving algorithms beating brute-force for some natural problems with mixed constraints, but we are still far from a general picture of how to obtain fast exponential algorithms for a combined template of constraints when each constraint type does admit such non-trivial algorithms. In this context, tackling the combination of k-CNF formulas and linear equations is a good starting point, and one that could hopefully spur a more systematic study in the future. There have been a few investigations [15, 17, 8, 16] into the fine-grained complexity of CSPs via the algebraic approach based on (partial) polymorphisms. This theory has developed the tools to compare the optimal exponents of different constraint types, identifying for instance the "easiest" NP-hard CSP within some classes. However, with the exception of [4], polymorphisms have not been leveraged to design fast exponential algorithms with competitive exponents.

1.1 Hardness results

Since k-Sub-Sat is a generalization of k-Sat, k-Sub-Sat inherits all the intractability results of k-Sat for $k \geqslant 3$. This leaves the interesting case of 2-Sub-Sat. This turns out to be much harder than the polynomial time solvable 2-Sat. We establish the following, showing not just hardness (even for FPT algorithms) of the exact version, but also a tight inapproximability

for the approximation version (even on satisfiable instances). The proofs are based on short, simple reductions, once an appropriate problem to reduce from is chosen.[1] The W[1]-hardness answers a question posed in [3] on the fixed-parameter complexity of 2-SAT with a global modular constraint, parameterized by the modulus.

▶ **Theorem 1.**

1. 2-*SUB-SAT is* NP*-hard. It is further* W[1]*-hard when parameterized by the co-dimension of the affine space A in which we seek a satisfying assignment.*
2. *Given a satisfiable instance of* 2-*SUB-SAT, it is* NP*-hard to find an assignment in the input space A that satisfies more than $3/4 + \epsilon$ of the 2SAT clauses, for any $\epsilon > 0$.*

1.2 Algorithmic results

Analogous to seeking k-SAT algorithms faster than brute-force, we investigate fast exponential time algorithms for k-SUB-SAT that beat the naive brute-force $2^{\dim(A)}$ time algorithm, where $A \subseteq \mathbb{F}_2^n$ is the subspace in which we seek a solution. Algorithms for k-SAT have received much attention and are central to the burgeoning field of fast exponential-time algorithms. The algorithmic theory is closely connected to fixed parameter tractability and parameterized complexity [11, 10]. The accompanying hardness theory [13, 14], based on the exponential-time hypothesis (ETH) and the strong exponential-time hypothesis (SETH), is a sanity check to the quest for faster algorithms for k-SAT and other NP-complete problems.

There are several interesting k-SAT algorithms with running time $O^*(2^{n(1-\Theta(1/k))})$.[2] We only mention two significant algorithms from among these: one by Paturi, Pudlak, Zane [21] and another due to Schöning [23]. Both algorithms are simple to describe with delightfully clever and elegant analyses. The PPZ algorithm considers variables in a random order, and gives each a random value unless its value is forced by a clause and previously set values. It achieves a running time of $O^*(2^{n(1-1/k)})$. Schöning's algorithm starts with a random assignment and in each step fixes an unsatisfied clause by flipping the value of a random one of its variables. It achieves a running time of $O^*((2-2/k)^n)$.

Given that k-SUB-SAT generalizes k-SAT, it is natural to seek exponential algorithms with similar running times for k-SUB-SAT. For SUB-SAT with input space $A \subseteq \mathbb{F}_2^n$, the brute-force algorithm in fact runs in time $O^*(2^{\dim(A)})$. A natural question is whether we can get similar improvements in the exponent of the $O^*(2^{\dim(A)})$ running time.

An algorithm [18] with running time about $O^*(2^{r(1-1/5k)})$ is known for checking satisfiability of a collection of arbitrary degree k polynomial equations in r variables: Let $P_i \in \mathbb{F}_2[x_1, x_2, \ldots, x_r]$, $1 \leqslant i \leqslant m$, be polynomials over the field $\mathbb{F}_2$. Following [18], the POLY-EQS problem is solving the system of polynomial equations $P_i = 0, 1 \leqslant i \leqslant m$ over $\mathbb{F}_2$: to check if there exists a solution in $\mathbb{F}_2^r$ and compute one if it exists. When P_i are all of degree bounded by k we denote this special case by k-POLY-EQS. The k-POLY-EQS problem generalizes k-SUB-SAT by the following easy transformation: Suppose the subspace A where we seek a satisfying assignment is r dimensional. Then we can express the i^{th} clause in the k-SUB-SAT instance as a disjunction of k affine linear forms in r variables: $C_i = (\ell_{i,1} \vee \ell_{i,2} \vee \cdots \vee \ell_{i,k})$. We define the corresponding polynomial $P_i = \prod_{j=1}^{k}(\ell_{i,j} + 1)$. Now, the k-SUB-SAT instance is satisfiable iff the k-POLY-EQS instance $P_i = 0, 1 \leqslant i \leqslant m$ has a solution in $\mathbb{F}_2^r$.

[1] The NP-hardness would also follow from Schaefer's dichotomy theorem for Boolean CSP [22], though that is an overkill hammer for this result.

[2] The notation $O^*(f(n))$ for running time bounds suppresses polynomial factors.

The algorithm [18] is a novel application of the Razborov-Smolensky "polynomial method," originally developed as a lower bound technique, used to define low-degree probabilistic polynomials for approximating the OR gate. The same idea allows for replacing a system of polynomial equations by a single probabilistic polynomial (without significant increase in degree), followed by a partial table lookup search. The article [18] presents more general results applicable to all finite fields $\mathbb{F}_q$. Recently, in [9], the running time for the case of $\mathbb{F}_2$ has been improved to $O^*(2^{r(1-1/2k)})$ by a refinement of the search method in [18].

Since k-Sub-Sat is a special case of solving a system of polynomial equations over $\mathbb{F}_2$, it raises the natural question of improving the running time further to match the $O^*(2^{r(1-1/k)})$ running time of the PPZ randomized algorithm for k-Sat. We are only able to achieve this speed-up in some special cases. However, on the positive side, our algorithms turn out to be *polynomial space bounded*, unlike the polynomial equations based method which requires exponential space [18, 9].

1.2.1 Algorithms for 2-Sub-Sat

For 2-Sub-Sat a simple deterministic branch-and-bound algorithm achieves a running time of $O^*(3^{r/2})$ where r is the dimension of the subspace A. We can improve on this with a randomized branching strategy to a running time of $O^*(1.5^r)$. This improves over the randomized $O^*(1.6181^r)$ algorithm given by the polynomial method [9] for solving a system of quadratic equations over $\mathbb{F}_2$. There is also a simple deterministic branching algorithm with $O^*(((1+\sqrt{5})/2)^r)$ running time for 2-Sub-Sat. This is based on the same branching strategy for k-Sat [19, Theorem, pp. 295] with its running time governed by the generalized Fibinacci numbers.

When $\dim(A) = n - t$, we can adapt the algorithm from [3, Algorithm 4.1] (for solving 2-SAT with a single abelian group constraint) to obtain an $O^*(\binom{n}{\leqslant t})$ time algorithm. [3]

The result of Theorem 1 shows that this problem is not in FPT parameterized by the co-dimension t, answering a question posed in [3] on whether 2-SAT with a global abelian group constraint might be fixed-parameter tractable, parameterized by the group size. More generally, the work [3] systematically studied the effect of a global modular constraint on the complexity of Boolean constraint satisfaction problems, exposing many interesting phenomena and connections.

Balancing the two running times of $O^*(1.5^r)$ and $O^*(\binom{n}{n-r})$ algorithm when $r \geqslant n/2$ (the exponents of the two bounds become equal at $r = (1-\eta)n$ for $\eta \approx 0.115816$) yields a $O^*(1.4312^n)$ time randomized algorithm for 2-Sub-Sat on n variables. The following records these results.

▶ **Theorem 2.** *There is a randomized $O^*(1.5^r)$ algorithm for 2-Sub-Sat where r is the dimension of the input space, as well a deterministic $O^*(\binom{n}{\leqslant t})$ time algorithm where t is the co-dimension. Together, these imply a randomized $O^*(1.4312^n)$ time algorithm as a function of the number n of variables.*

1.2.2 Algorithms for k-Sub-Sat

We explore combinatorial algorithms for k-Sub-Sat based on the notion of *critical variables* (which was introduced in [21] and plays an important role in their satisfiability algorithm). Let Φ be a satisfiable CNF formula in n variables $x_i, i \in [n]$, and let $\bar{a} \in \mathbb{F}_2^n$ be a satisfying assignment.

[3] For nonnegative integers n, t, the notation $\binom{n}{\leqslant t}$ stands for $\sum_{i=0}^{t} \binom{n}{i}$.

▶ **Definition 3** ([21]). *We say x_i is a* critical variable *for $\bar{a}$ with respect to Φ if the assignment $\bar{a} + e_i$ falsifies Φ, where e_i is the i^{th} elementary vector with 1 in the i^{th} coordinate and zero elsewhere (so $\bar{a} + e_i$ is just $\bar{a}$ with x_i flipped). If the formula Φ is clear from context, we simply say that x_i is a critical variable for assignment $\bar{a}$.*

The key idea in our combinatorial algorithms is *plucking* of non-critical variables based on the following simple observation: if Φ is an n-variate CNF formula and $\bar{a}$ is a satisfying assignment such that variable x_i is non-critical for it, then the formula Φ' obtained by plucking x_i (i.e., dropping all occurrences of x_i and its complement from Φ) remains satisfiable with $\bar{a}' \in \mathbb{F}_2^{n-1}$ as a satisfying assignment, where $\bar{a}'$ is obtained from $\bar{a}$ by dropping the i^{th} coordinate.

The important property of Φ' is that given any satisfying assignment for Φ' we can set x_i to either 0 or 1 to recover a satisfying assignment for Φ. This facilitates searching for a satisfying assignment in an affine space A: if the plucked variable x_i occurs in a linear constraint defining A then we can drop that linear constraint while seeking a satisfying assignment for Φ', because that linear constraint can always be satisfied by choosing the right value of x_i which still remains overall a satisfying assignment for Φ. Based on this idea we obtain the following algorithms for k-Sub-Sat:

- The first result here is a randomized $O^*(\binom{n}{t}2^{n-n/k})$ time algorithm for k-Sub-Sat where $t = \text{codim}(A)$. This algorithm is essentially governed by the running time of the PPZ satisfiability algorithm [21] combined with an iterative "search and pluck" operation to remove t non-critical variables from the t linear equations defining A. This running time is superior to the $O^*(2^{r-r/2k})$ time randomized algorithm based on solving polynomial equations for small values of $t = o(n)$.
- The second result is a general randomized $O^*(2^{n-n/2k+n/2k^2})$ time algorithm for k-Sub-Sat, nearly matching the $\approx 2^{r-r/2k}$ run time of the polynomial equations algorithm [9, 18] for r close to n. It again uses the PPZ satisfiability algorithm as a subroutine combined with simple applications of the plucking step: if the number of critical variables is fewer than $n/2$, it randomly guesses and plucks non-critical variables. This algorithm does not need to look at the linear equations defining A. In fact, it works for any Boolean constraint $C(x_1, x_2, \ldots, x_n)$ (replacing membership in the affine space A) with a polynomial-time algorithm that takes a partial assignment and extends it to an assignment that satisfies C. For example, C can be a HORN or dual HORN formula.
- It is pleasing to note that we can apply the idea of plucking non-critical variables to 2-Sub-Sat and obtain an $O^*(\binom{n}{\leqslant t})$ deterministic algorithm (cf. [3]), where $t = \text{codim}(A)$. Exploiting the structure of 2-CNF formulas, we can find the non-critical variables efficiently.

▶ **Theorem 4.** *The k-Sub-Sat problem admits two randomized algorithms, one running in time $O^*(2^{n-n/2k+n/2k^2})$, and another running in $O^*(\binom{n}{t}2^{n-n/k})$ when the input subspace has co-dimension $t \leqslant n/2$.*[4] *Both algorithms use space bounded by a polynomial in n.*

▶ Remark 5. Satisfiability algorithms based on the switching lemma (which converts k-CNF to decision trees of moderate term size and number of terms) are known in the literature (e.g., see [12]). We can easily adapt this algorithm to solve k-Sub-Sat, because once we have a decision tree for the underlying k-CNF formula, for the k-Sub-Sat instance each path of the decision tree will give rise to a system of linear equations over $\mathbb{F}_2$. For each path, therefore,

[4] Of course, there is also a trivial $O^*(2^{n-t})$ time brute force algorithm.

we can even count the number of satisfying assignments. Counting over all the paths of the decision tree gives the total number of satisfying assignments for the k-Sub-Sat instance in randomized time $O^*(2^{n(1-1/c\cdot k)})$ for some suitable large constant $c > 0$. Furthermore, the algorithm is also polynomial space-bounded. In terms of running time, however, it is a much weaker bound in comparison to [18] or even the algorithms of Theorem 4. In this context, we note that for #k-Sat there is a *deterministic* $O^*(2^{n(1-1/c\cdot k)})$ time algorithm based on the polynomial method (albeit using exponential space) [6]. We do not know of any such deterministic algorithm for counting satisfying assignments to k-Sub-Sat.

Finally, motivated by the (unbounded CNF) Sub-Sat problem, we revisit the general problem solving a system of polynomial equations $p_i = 0, 1 \leqslant i \leqslant m$ over $\mathbb{F}_2$, where $m = O(n)$, where each p_i is given by an arithmetic circuit of poly(n) degree. In the case when each p_i has small degree irreducible factors, we get a $2^{r(1-\alpha)}$ time randomized algorithm, where α depends on the number of equations m and the degree bound on the irreducible factors (Theorem 25).

1.3 Equivalent and related problems to Sub-Sat

Recall the USA problem: Given a collection of affine subspaces $A_1, A_2, \ldots, A_m \subseteq \mathbb{F}_2^n$ (where each A_i is given by a bunch of affine linear equations over $\mathbb{F}_2$) the problem is to determine if there is a point $x \in \mathbb{F}_2^n \setminus \bigcup_{i=1}^m A_i$.

Clearly, the complement $\mathbb{F}_2^n \setminus \bigcup_{i=1}^m A_i$ is expressible as an AND of ORs of affine linear forms $\oplus_{i\in S} x_i + b$, $b \in \{0, 1\}$. Thus, USA is clearly reducible to Sub-Sat. The converse reduction is also easy: given a CNF formula Φ and an affine subspace $A \subseteq \mathbb{F}_2^n$ we first convert it to an AND of ORs of affine linear forms. An assignment $x \in A$ satisfies Φ if and only if it satisfies $C_1 \wedge C_2 \wedge \cdots \wedge C_m$, where each clause C_i is an OR of affine linear forms. The set A_i of satisfying assignments of the complement $\overline{C_i}$ is an affine subspace of $\mathbb{F}_2^n$, and Φ is satisfiable by $x \in A$ if and only if $x \in \mathbb{F}_2^n \setminus \bigcup_{i=1}^m A_i$.

For the equivalence to PAF-Sat, suppose $\Phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$, where each clause C_i is an OR of affine linear forms $C_i = \vee_{j=1}^t L_{ij}$. As already discussed in Section 1.2, the assignment $x \in \mathbb{F}_2^n$ satisfies C_i if and only if it satisfies the polynomial equation $\prod_{j=1}^m (L_{ij} + 1) = 0$. Thus, the satisfiability of Φ is reducible to a system of m polynomial equations $p_i = 0$, where each p_i is a product of affine linear forms. The converse reduction is also easy which we omit.

Organization of the paper

We present the results in a different order than in the introduction. In Section 2 we first present the algorithms for k-Sub-Sat and then for 2-Sub-Sat. In Section 3 we present our hardness results for 2-Sub-Sat. Finally, in Section 4 we present the algorithm for Poly-Eqs for $O(n)$ equations $p_i = 0$, where each p_i has unrestricted degree but constant-degree irreducible factors.

For reasons of space, all proofs are skipped in the extended abstract; a full version of the paper is available on arXiv [1].

2 Algorithmic results for k-Sub-Sat

As mentioned in the introduction, the k-Sub-Sat problem seems intermediate in difficulty, between k-Sat and the problem k-Poly-Eqs of solving a system of degree-k polynomial equations over $\mathbb{F}_2$. The latter problem has an $O^*(2^{r(1-1/2k)})$ time algorithm [18, 2, 9], which yields an $O^*(2^{r(1-1/2k)})$ time algorithm for k-Sub-Sat, where $r = \dim(A)$.

Ideally, we would like an algorithm for k-Sub-Sat with run time $O^*(2^{r(1-1/k)})$, with savings in the exponent similar to that of the PPZ algorithm [21] for k-Sat.

We present some algorithms in this direction: For 2-Sub-Sat there is a simple $O^*(1.5^r)$ time randomized algorithm which improves on the $O^*(2^{r(1-1/2k)})$ bound for $k = 2$. For a special case of k-Sub-Sat, when $r = \dim(A)$ is close to the number of variables n, we are able to adapt the PPZ algorithm to essentially get an $O^*(2^{r(1-1/2k)})$ time algorithm. Writing $t = n - r = \mathrm{codim}(A)$, we can even obtain an $O^*(\binom{n}{\leqslant t} \cdot 2^{n(1-1/k)}$ time algorithm for the problem, also based on the PPZ satisfiability algorithm, which yields the desired $1/k$ savings in the exponent for small t.

2.1 An $O^*(\binom{n}{t} \cdot 2^{n(1-1/k)})$ time randomized algorithm: co-dimension t case

As outlined in Section 1.2, the algorithm will use the PPZ satisfiability algorithm [21] as a subroutine, combined with variable plucking steps to solve k-Sub-Sat in randomized time $O^*(\binom{n}{t} \cdot 2^{n(1-1/k)})$, when $\mathrm{codim}(A) = t$. In particular, for $\mathrm{codim}(A) = o(n)$ the algorithm has run time $O^*(2^{n(1-1/k+o(1))})$.

The variable plucking is based on analyzing the critical variables for a solution $\bar{a} \in \mathbb{F}_2^n$ of a given k-Sub-Sat instance (Φ, A), depending on whether or not they occur in the linear equations defining A.

For an instance (Φ, A) we partition the variables into two sets

$$\{x_i \mid i \in [n]\} = V_{\mathrm{in}} \sqcup V_{\mathrm{out}},$$

where V_{in} is the subset of variables that have nonzero coefficient in at least one of the t linear equations defining A, and V_{out} is the remaining set of variables. By abuse of notation, we will also treat $V_{\mathrm{in}} \sqcup V_{\mathrm{out}}$ as a partition of the index set $[n]$. We consider the following two cases.

Case 1. Suppose (Φ, A) has the property that *for every solution* $\bar{a} \in \mathbb{F}_2^n$ each variable in V_{in} is critical for $\bar{a}$ w.r.t Φ. There is no variable plucking required in this case. It only involves the application of the PPZ satisfiability algorithm on Φ and checking that the assignment found belongs to A. We need the following lemma which is analogous to [21, Lemma 4]. The proof of the lemma is by an induction argument like in [21].

▶ **Lemma 6.** *Let S be a nonempty subset of $\mathbb{F}_2^n$. For each $\bar{a} \in S$, let $I_{out}(\bar{a}) = \{i \in V_{out} \mid \bar{a} + e_i \notin S\}$, where e_i is the i^{th} elementary vector. Then we have*

$$\sum_{\bar{a} \in S} 2^{|I_{out}(\bar{a})| - |V_{out}|} \geqslant 1. \tag{1}$$

Now, let $\bar{a} \in \mathbb{F}_2^n$ be some solution of the k-Sub-Sat instance (Φ, A). Then, by the assumption of Case 1 and the preceding discussion $\bar{a}$ has $|V_{\mathrm{in}}| + |I_{\mathrm{out}}(\bar{a})|$ critical variables w.r.t Φ.

Following the analysis in [21], if we now run one iteration of the PPZ algorithm on the instance Φ, the probability that $\bar{a}$ is output is at least

$$\frac{1}{n^2} \cdot 2^{-n + (|V_{\mathrm{in}}| + |I_{\mathrm{out}}(\bar{a})|)/k}.$$

Let $S \subset \mathbb{F}_2^n$ denote the subset of solutions to the instance (Φ, A). Summing up over all $\bar{a} \in S$, the probability that some solution $\bar{a}$ is output is given by

$$\begin{aligned}\sum_{\bar{a}\in S} \frac{1}{n^2} \cdot 2^{-n+(|V_{\text{in}}|/k+|I_{\text{out}}(\bar{a})|/k)} &= \frac{1}{n^2} 2^{-n+n/k} \cdot \sum_{\bar{a}\in S} 2^{(-|V_{\text{out}}|/k+|I_{\text{out}}(\bar{a})|)/k} \\ &\geqslant \frac{1}{n^2} 2^{-n+n/k} \cdot \sum_{\bar{a}\in S} 2^{(-|V_{\text{out}}|+|I_{\text{out}}(\bar{a})|)} \geqslant \frac{1}{n^2} 2^{-n+n/k} ,\end{aligned}$$

where the last step uses Lemma 6. This finishes the analysis of Case 1.

▶ Remark 7. Notice in the probability analysis that S is the set of solutions to (Φ, A) and not all solutions to Φ. The crucial property that for every $\bar{a} \in S$, each variable in V_{in} is critical w.r.t Φ yields that there are $|V_{\text{in}}| + |I_{\text{out}}(\bar{a})|$ critical variables for $\bar{a}$ w.r.t Φ. Intuitively, as the variables in V_{out} do not occur in the linear equations, the PPZ algorithm when run on Φ will be able to deterministically set, on average, $|I_{\text{out}}(\bar{a})|/k$ many of the critical variables in V_{out} without any interaction with the linear equations defining A.

Case 2. We now consider the case when not all variables in V_{in} are critical to all solutions to (Φ, A). We will show that there is a subset of at most t variables in V_{in} that can be plucked from Φ and reduce the transformed instance to Case 1. We will argue that the algorithm can do an exhaustive search for this subset of V_{in} of size at most t.

▶ **Lemma 8.** *In the k-Sub-Sat instance (Φ, A), let $Bx = b$ be the system of t linear equations defining A. Suppose variable x_1 occurs in the first equation $\sum_{j=1}^{n} B_{1j}x_j = b_1$ (i.e., $B_{11} \neq 0$). Further, suppose x_1 is not critical for some solution to (Φ, A). Let Φ' be the formula obtained by plucking x_1 from Φ. Let A' be the affine space of co-dimension $t-1$ defined by dropping the first linear equation $\sum_{j=1}^{n} B_{1j}x_j = b_1$* after eliminating *$x_1$ from the other linear equations by row operations. Then (Φ', A') is satisfiable and* any solution *$\bar{a}'$ to (Φ', A') can be extended to a solution $\bar{a}$ of (Φ, A).*

Lemma 8 describes a pluck/eliminate step applied to the non-critical variable x_1: namely, pluck x_1 from Φ and eliminate it from the equations describing A.

Clearly, for some sequence of $s \leqslant t$ pluck/eliminate steps applied successively transforms (Φ, A) to (Φ_s, A_s) for which Case 1 holds. Since we do not have an efficient test for checking non-criticality, the algorithm has to do an exhaustive search for the sequence of s variables to pluck/eliminate. The number of variable sequences to consider is bounded by n^t. However, as we argue in the next claim, it suffices to consider each unordered subset U of size $s \leqslant t$ variables and apply pluck/eliminate steps to its variables in the natural order $x_1, \ldots, x_n$. Thus, we can bound the exhaustive search to $\binom{n}{\leqslant t}$ subsets of variables. Let (Φ_U, A_U) be the resulting instance after pluck/eliminate applied to variables in U in the natural order.

▶ **Lemma 9.** *Let (Φ, A) be a satisfiable instance of k-Sub-Sat with* $\operatorname{codim}(A) = t$. *There is a subset U of variables of size at most t, such that (Φ_U, A_U) is a satisfiable Case 1 instance of k-Sub-Sat.*

The $O^*(\binom{n}{\leqslant t} \cdot 2^{n-n/k})$ time Algorithm. On input (Φ, A), the algorithm proceeds as follows:

For each subset $U \subset V_{\text{in}}$ of size at most t do the following:

1. Pluck the variables in U from Φ to obtain Φ_U.

2. For each variable $x_i \in U$ (in any order): pick some equation in which x_i occurs; remove x_i from other equations by adding the picked equation to it; drop the picked equation from the system.
3. Run the PPZ algorithm on the resulting instance (Φ_U, A_U) as if Case 1 were applicable. More precisely, run PPZ on Φ_U for $O^*(2^{n-n/k})$ steps; for each solution obtained, if it satisfies A_U then output an extension of it to a solution to (Φ, A) and exit,[5] else continue the for-loop for the next choice of subset U.

To see the correctness, suppose (Φ, A) is satisfiable. By Lemma 9, for some choice of U with $|U| \leqslant t$, (Φ_U, A_U) is a Case 1 instance. Hence, the PPZ satisfiability algorithm will output a solution to (Φ_U, A_U) in time $O^*(2^{n-n/k})$ with high probability. This solution can be uniquely extended to a solution to (Φ, A) using the linear equations.

We have thus shown the following.

▶ **Theorem 10.** *There is a randomized $O^*(\binom{n}{t} \cdot 2^{n-n/k})$ time algorithm for k-Sub-Sat for subspaces of co-dimension t. In particular, for $t = o(n)$ we have a randomized $O^*(2^{n(1-1/k+o(1))})$ time algorithm.*

2.2 An $O^*(2^{n-n/2k+n/2k^2})$ time PPZ-based algorithm for k-Sub-Sat

Let (Φ, A) be a k-Sub-Sat instance. Our objective is a randomized algorithm with run time $2^{n-(1-\nu)n/k}$ for as small an ν as possible (ideally, tending to zero).

To this end, we can first apply Valiant-Vazirani Lemma [27] to increase the number of constraints (thereby reducing the rank of A) and getting an instance (Φ, A') such that Φ has a unique solution in A' with high probability (i.e., inverse polynomial probability as guaranteed by Valiant-Vazirani).

If $\dim(A') \leqslant n - (1-\nu)n/k$ we can brute force search in A' in deterministic time $2^{\dim(A')} \leqslant 2^{n-(1-\nu)n/k}$. Thus, we can assume that $\dim(A') = n - t$ and A' is the solution space of $t < (1-\nu)n/k$ independent affine linear equations.

Let now $\bar{a} \in \mathbb{F}_2^n$ be the unique solution to the k-Sub-Sat instance (Φ, A'). We partition the variable set into $V_{\text{in}} \sqcup V_{\text{out}}$ as before.

▷ Claim 11. Every variable in V_{out} is critical for the satisfying assignment $\bar{a}$ of Φ.

Proof. Suppose $x_i \in V_{\text{out}}$ is not critical for $\bar{a}$. Then $\bar{a} + e_i$ is also a satisfying assignment for Φ. Moreover, since x_i does not occur in V_{in}, $\bar{a} + e_i$ satisfies the linear equations defining A'. Hence $\bar{a} + e_i$ is a solution to (Φ, A') contradicting the uniqueness of $\bar{a}$.

The variable plucking algorithm. If $\bar{a}$ has more than $(1-\nu)n$ many critical variables (ν to be fixed in the analysis) then by running the PPZ satisfiability algorithm [21] for $O^*(2^{n-(1-\nu)n/k})$ iterations we will find it with high probability.

Otherwise, there are more than νn many variables in V_{in} that are *not critical* for Φ at $\bar{a}$.

1. Repeat the following two steps at most t times.
2. (The plucking step) Randomly pluck a variable x_i from V_{in} and drop it from the formula Φ to obtain its *shrinking* Φ_1. Take a linear equation $\ell = b$ in which x_i occurs. By row operations eliminate x_i from all other linear equations in which x_i occurs and then drop the equation $\ell = b$. Let the affine space described by the new set of at most $t-1$ linear

[5] From a solution to (Φ_U, A_U) we can reconstruct the solution to (Φ, A) as the values to variables in U are uniquely determined via the linear equations from the values to the other variables.

equations be A_1. We claim that (Φ_1, A_1) also has a unique solution $\bar{a}_1$ (obtained from $\bar{a}$ by dropping the i^{th} coordinate).

3. Let $n_1 = n - 1$. Run the PPZ algorithm for $2^{n_1-(1-\nu)n_1/k}$ time on Φ_1. If we do not find the unique solution $\bar{a}_1$ then repeat the plucking step.

At the end of t successful plucking steps we are left with a k-SAT instance Φ_t with a unique solution (the subspace A_t is $\mathbb{F}_2^n$) and PPZ will find that solution from which we can compute $\bar{a}$ by recovering the unique values of the plucked variables using the linear equations.

Analysis. At the j^{th} iteration of the plucking step, the probability that all j steps pluck off non-critical variables is at least ν^j. Thus, the running time of the search for unique solutions for the (Φ_j, A_j) over all t steps is bounded by $\sum_{j=0}^{t} O^*(\frac{1}{\nu^j} \cdot 2^{n_j-(1-\nu)n_j/k})$.

Letting $\alpha = 2^{1-(1-\nu)/k}$ and noting that $n_j = n - j$ we can rewrite and bound the above sum as

$$\begin{aligned} O^*(2^{n-(1-\nu)n/k}) \cdot \sum_{j=0}^{t} \frac{1}{\nu^j \cdot \alpha^j} &\leqslant O^*(2^{n-(1-\nu)n/k}) \cdot t \cdot \frac{1}{\nu^t \cdot \alpha^t} \\ &\leqslant O^*(2^{n-(1-\nu)n/k}) \cdot t \cdot \left(\frac{1}{2\nu}\right)^{(1-\nu)n/k} \cdot 2^{(1-\nu)n/k^2}, \end{aligned}$$

as the sum $\sum_{j=0}^{t} \frac{1}{\nu^j \cdot \alpha^j}$ is bounded by $t\frac{1}{\nu^t \cdot \alpha^t}$ for $\nu\alpha < 1$ and $t \leqslant (1-\nu)n/k$.

The overall running time of the algorithm is, therefore, $O^*(2^{n-n/k}) \cdot 2^{\nu n/k} \cdot \left(\frac{1}{2\nu}\right)^{(1-\nu)n/k} \cdot 2^{(1-\nu)n/k^2}$, which is minimized at $\nu = 1/2$ as we argue below, and is given by $O^*(2^{n-n/2k+n/2k^2})$.

▶ Remark 12 (Extension beyond linear-algebraic constraints). We note some aspects about the algorithm and explain its adaptation to the more general setting of k-CNF satisfiability in the presence of a global boolean constraint $C(x_1, x_2, \ldots, x_n)$ with the property that given a partial assignment to the variables x_i we can extend the assignment to the remaining variables that satisfies the constraint C, if such an extension exists. We set $\nu = 1/2$ and $t = n/2k$. Note that the algorithm need not partition the variables into V_{in} and V_{out}. If there are over $n/2$ non-critical variables, the algorithm can "obliviously" pluck one with probability $1/2$. Oblivious in the sense that it does not need to see the constraint C. After $t = n/2k$ plucking steps, there are at most $n - n/2k$ remaining variables. We add a final step to the algorithm which is a brute-force search over all $2^{n-n/2k}$ assignments to the remaining variables. For each assignment to these that satisfies Φ_t we can check, in polynomial time, if there is an extension to it that satisfies C. This search will succeed for the unique solution $\bar{a}$. An interesting example for constraint C would be HORN formulas. As clause size is unrestricted in HORN formulas, notice that neither a direct application of the PPZ satisfiability algorithm, nor an application of the polynomial equations algorithms would give constant savings in the exponent for the running time bound.

More generally, call a Boolean constraint $C(x_1, x_2, \ldots, x_n)$ $T(n)$-easy if there is a $T(n)$ time-bounded algorithm that searches for a satisfying extension of a given partial assignment to the variables x_i.

▶ **Theorem 13.** *There is a randomized $O^*(2^{n-n/2k+n/2k^2} \cdot T(n))$ time algorithm that takes any k-CNF formula and a $T(n)$-easy boolean constraint $C(x_1, x_2, \ldots, x_n)$ as input and computes a satisfying assignment for the formula and C.*

▶ **Corollary 14.** *There is a randomized $O^*(2^{n-n/2k+n/2k^2})$ time algorithm for k-SUB-SAT.*

2.3 An $O^*(1.5^r)$ time algorithm for 2-Sub-Sat

▶ **Theorem 15.** *Given a* 2-Sub-Sat *instance* (Φ, A)*, where* Φ *is a* 2-CNF *formula and* $A \subset \mathbb{F}_2^n$ *is an* r*-dimensional affine subspace given by linear equations, there is a randomized* $O^*(1.5^r)$ *time algorithm to check if* Φ *has a satisfying assignment in* A *and if so to compute it.*

▶ Remark 16. The run time of $O^*(1.5^r)$ that we obtain improves on the polynomial equations based algorithms, where for $k = 2$ the best run time so far is $O^*(1.618^r)$ [9]. For $k = 3$ a similar randomized branching strategy gives an algorithm with run time $O^*((7/4)^r)$. For larger k the run time degrades to $O^*((2 - 1/2^{k-1})^r)$. This running time bound is obtained similarly as for Theorem 15: fix a satisfying assignment $\bar{a}$ of the k-Sub-Sat instance. For a clause $(\ell_1 \vee \ell_2 \vee \cdots \vee \ell_k)$ of k linearly independent linear forms a random (nonzero) linear combination $\sum_{i=1}^{k} \alpha_i \ell_i$ evaluates to 1 at $\bar{a}$ with probability exactly $\frac{2^{k-1}}{2^k-1}$.

2.4 2-Sub-Sat in a co-dimension t subspace

In this section we consider 2-Sub-Sat where we are seeking a solution in an affine space A such that $\mathrm{codim}(A) = t$.

Given a formula Φ we will identify a canonical satisfying assignment $\bar{a}$ for Φ based on which we will define critical variables. Since 2-Sat is in polynomial-time, we can detect non-critical variables in Φ w.r.t. $\bar{a}$ in polynomial time. Now the plucking step will try all the possible $\binom{n}{t}$ choices of plucking non-critical variables, recalling that a non-critical variable plucked from a linear constraint defining A allows us to drop that constraint.

▶ **Theorem 17.** *There is an* $O^*(\binom{n}{t})$ *time deterministic algorithm for checking if a* 2-Sub-Sat *instance* (Φ, A) *is satisfiable where the affine space* A *has co-dimension* t*.*

3 Hardness results

In this section we prove our hardness results for subspace satisfiability. Since k-Sat itself is NP-hard for $k \geqslant 3$, so is k-Sub-Sat for $k \geqslant 3$. So we focus on the case $k = 2$.

3.1 NP-hardness of 2-Sub-Sat

While 2-Sat is polynomial time solvable, the following theorem shows that 2-Sub-Sat is NP-hard. Note that this follows from Schaefer's dichotomy theorem for Boolean CSP as the combination of 2-Sat constraints and linear equations (even with 3 variables per equation) is not one of the six tractable cases, and thus NP-hard. Below we give a direct proof based on a simple reduction.

▶ **Theorem 18.** 2-Sub-Sat *is NP-hard.*

3.2 W[1]-hardness of 2-Sub-Sat parameterized by co-dimension

We now strengthen the hardness result of Theorem 18 and show that 2-Sub-Sat is unlikely to even be fixed-parameter tractable when parameterized by the co-dimension t of the subspace in which we seek a satisfying assignment to the 2CNF formula. On the other hand, recall that (as shown in [3] and also Section 2.4), for fixed co-dimension t, 2-Sub-Sat can be solved in polynomial time. Our W[1]-hardness answers (in the negative) a question posed in [3] on whether 2-Sat with a single modular constraint modulo M is fixed-parameter tractable when parameterized by M (they gave an algorithm with complexity $n^{O(M)}$).

▶ **Theorem 19.** *Consider the* 2-Sub-Sat *where the input subspace within which one has to satisfy the* 2-Sat *formula has co-dimension* t*. Parameterized by* t*,* 2-Sub-Sat *is* W[1]*-hard.*

3.3 Approximability of Max-2-Sub-Sat

Given the hardness of deciding exact satisfiability of 2-Sub-Sat instance, we now turn to approximate satisfiability. In the Max-2-Sub-Sat problem, the goal is to satisfy the maximum number of 2SAT clauses with an assignment that belongs to the input affine space A. Thus, the affine constraints are treated as hard constraints. We allow clauses of width 1. If unary clauses are disallowed in the 2CNF formula, and each clause involves exactly two distinct variables, we call the problem Max-E2-Sub-Sat.

3.3.1 Easy approximation algorithms

We can assume that no variable is forced to 0 or 1 by the affine space A, since if that happens we can just set and remove that variable and work on the reduced instance. If we pick a random assignment from A, it will satisfy at least $1/2$ of the clauses of the 2CNF formula in expectation, and in fact at least an expected fraction $3/4$ of the clauses when each clause involves two distinct variables. The algorithms are easily derandomized. For satisfiable instances of Max-2-Sub-Sat, one can find a $3/4$ approximate solution, as one can eliminate all the unary clauses, and add those conditions to the subspace inside which we want to find an assignment to the 2CNF formula. So we get the following trivial algorithmic guarantees.

▶ **Observation 20.** *In polynomial time, one can get a factor* $1/2$ *approximate solution to instances of Max-2-Sub-Sat, a factor* $3/4$ *approximate solution to instances of Max-E2-Sub-Sat, and a factor* $3/4$ *approximate solution to satisfiable instances of Max-2-Sub-Sat.*

We will now show that all the above guarantees are best possible, with matching NP-hardness results.

3.3.2 Tight inapproximability via simple reductions

For the hardness results and rest of the section, it is convenient to work with the PAF-Sat formulation of Sub-Sat. The Max-LIN2 problem, of maximizing the number of satisfied equations in a system of affine equations mod 2, trivially reduces to Max-2-PAF-Sat (with each equation being degree 1 instead of degree 2). By Håstad's seminal tight inapproximability for Max-LIN2, we have the following.

▶ **Observation 21.** *For any* $\epsilon > 0$*, Max-2-PAF-Sat (and thus Max-2-Sub-Sat) is NP-hard to approximate within a factor of* $(1/2 + \epsilon)$*, and this holds for almost satisfiable instances that admit an assignment satisfying a fraction* $(1 - \epsilon)$ *of equations.*

We also get a tight hardness (matching Observation 20) for the Max-E2-Sub-Sat or equivalently when each polynomial equation is the product of exactly two (linearly independent) affine forms.

▶ **Lemma 22.** *For any* $\epsilon > 0$*, Max-E2-PAF-Sat is NP-hard to approximate within a factor of* $(3/4+\epsilon)$*, and this holds for almost satisfiable instances that admit an assignment satisfying a fraction* $(1 - \epsilon)$ *of equations.*

3.3.3 Inapproximability for satisfiable instances

The above inpproximability results do not apply to *satisfiable* instances of 2-Sub-Sat. They are obtained by reductions from linear equations whose exact satisfiability can be easily checked. We now prove that approximating Max-2-Sub-Sat doesn't get easier on satisfiable instances.

▶ **Theorem 23.** *For every $\epsilon > 0$, it is NP-hard to approximately solve satisfiable instance of Max-E2-Sub-Sat within a factor of $3/4 + \epsilon$. That is, it is NP-hard to find, given as input a satisfiable instance of 2-Sub-Sat, an assignment satisfying a fraction $3/4 + \epsilon$ of the 2SAT constraints.*

4 System of polynomial equations over binary field: effect of reducibility

We now examine a special case of the problem of solving a system of polynomial equations over $\mathbb{F}_2$ studied in [18, 2, 9]. For motivating background, we recall according to the strong exponential time hypothesis (SETH) that Sat, that is n-variable CNF satisfiability of unrestricted clause width, cannot be essentially solved faster than 2^n time. However, Schuler [24] and Calabro et al [5] have shown the special case that sparse instances of Sat (with $c \cdot n$ clauses) can be solved in $O^*(2^{n(1-\alpha)})$ time, where α is a constant depending on the clause density c. It is natural to ask if there is an analogous result for Sub-Sat (satisfiability of conjunctions of unbounded disjunctions of affine linear forms). In this section we show a more general algorithmic result in the setting of systems of polynomial equations over $\mathbb{F}_2$.

Let $P_i \in \mathbb{F}_2[x_1, x_2, \ldots, x_n], 1 \leqslant i \leqslant m$ be polynomials over the field $\mathbb{F}_2$ as input instance to the Poly-Eqs problem. The problem is denoted k-Poly-Eqs when the degrees are bounded by k which generalizes k-Sub-Sat as already explained in the introduction.

The unrestricted degree case is significantly different, because we can easily combine the m equations into a single equation as follows. Define

$$P = 1 + \prod_{i=1}^{m} (1 + P_i).$$

Clearly, the system $P_i = 0, 1 \leqslant i \leqslant m$ has a solution iff $P = 0$ has a solution.

Thus, assuming SETH, there is no algorithm essentially faster than 2^n for solving $P = 0$.

▶ Remark 24. There is also the question of how the polynomials P_i are given as part of the input. If $\deg P_i \leqslant k$ for all P_i then we can in polynomial-time compute their sparse representation as a linear combination of the n^k many monomials of degree at most k. However, in the above reduction of combining the P_i into a single polynomial, P is a small arithmetic formula. In fact, for the case of Poly-Eqs we consider, where the instance is a system of equations $P_i = 0, 1 \leqslant i \leqslant m$ such that $m = O(n)$ and each P_i has constant degree irreducible factors, we can assume that the P_i are given as arithmetic circuits.

We now show that Poly-Eqs instances $P_i = 0, 1 \leqslant i \leqslant m$ can be solved faster than 2^n if m is linear in n and the irreducible factors of each P_i are of constant degree. This can be seen as a "polynomial equations" analogue of Schuler's Sat algorithm for spare instances with unrestricted clause width [24, 5]. We note that a different degree reduction method, based on a rank argument, is used in [18, Section 4] to solve systems of polynomial equations $p_i = 0$, where each p_i is given by a sum of product of affine linear forms.

▶ **Theorem 25.** *Let $P_i = 0, 1 \leqslant i \leqslant c \cdot n$, for a constant $c > 0$, be an instance of* Poly-Eqs*, such that the degree of each irreducible factor of each P_i is bounded by a constant b. There is a randomized algorithm for* Poly-Eqs *that runs in time $2^{n(1-\alpha)}$ for such instances, where $\alpha > 0$ is a constant that depends on c and b.*

References

1. Vikraman Arvind and Venkatesan Guruswami. Cnf satisfiability in a subspace and related problems, 2021. arXiv:2108.05914.
2. Andreas Björklund, Petteri Kaski, and Ryan Williams. Solving systems of polynomial equations over GF(2) by a parity-counting self-reduction. 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 26:1–26:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
3. Joshua Brakensiek, Sivakanth Gopi, and Venkatesan Guruswami. CSPs with global modular constraints: algorithms and hardness via polynomial representations. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC)*, pages 590–601, 2019.
4. Joshua Brakensiek and Venkatesan Guruswami. Bridging between 0/1 and linear programming via random walks. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing*, pages 568–577, 2019.
5. Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. A duality between clause width and clause density for SAT. In *21st Annual IEEE Conference on Computational Complexity (CCC 2006), 16-20 July 2006, Prague, Czech Republic*, pages 252–260. IEEE Computer Society, 2006.
6. Timothy M. Chan and R. Ryan Williams. Deterministic APSP, orthogonal vectors, and more: Quickly derandomizing Razborov-Smolensky. *ACM Trans. Algorithms*, 17(1):2:1–2:14, 2021.
7. Ruiwen Chen and Rahul Santhanam. Satisfiability on mixed instances. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, pages 393–402, 2016.
8. Miguel Couceiro, Lucien Haddad, and Victor Lagerkvist. Fine-grained complexity of constraint satisfaction problems through partial polymorphisms: A survey. In *2019 IEEE 49th International Symposium on Multiple-Valued Logic (ISMVL)*, pages 170–175, 2019.
9. Itai Dinur. Improved algorithms for solving polynomial systems over GF(2) by multiple parity-counting. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2550–2564. SIAM, 2021.
10. Rodney G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer Publishing Company, Incorporated, 2012.
11. J. Flum and M. Grohe. *Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series).* Springer-Verlag, 2006.
12. Russell Impagliazzo, William Matthews, and Ramamohan Paturi. A satisfiability algorithm for AC^0. 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 961–972. SIAM, 2012.
13. Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.
14. Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.
15. Peter Jonsson, Victor Lagerkvist, Gustav Nordh, and Bruno Zanuttini. Strong partial clones and the time complexity of SAT problems. *J. Comput. Syst. Sci.*, 84:52–78, 2017.
16. Peter Jonsson, Victor Lagerkvist, and Biman Roy. Fine-grained time complexity of constraint satisfaction problems. *ACM Trans. Comput. Theory*, 13(1):2:1–2:32, 2021.

17 Victor Lagerkvist and Magnus Wahlström. Which NP-hard SAT and CSP problems admit exponentially improved algorithms? *CoRR*, abs/1801.09488, 2018. `arXiv:1801.09488`.

18 Daniel Lokshtanov, Ramamohan Paturi, Suguru Tamaki, R. Ryan Williams, and Huacheng Yu. Beating brute force for systems of polynomial equations over finite fields. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2190–2202, 2017.

19 Burkhard Monien and Ewald Speckenmeyer. Solving satisfiability in less than 2^n steps. *Discret. Appl. Math.*, 10(3):287–295, 1985.

20 Mihai Patrascu and Ryan Williams. On the possibility of faster SAT algorithms. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1065–1075, 2010.

21 Ramamohan Paturi, Pavel Pudlák, and Francis Zane. Satisfiability coding lemma. In *38th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 566–574, 1997.

22 Thomas J. Schaefer. The complexity of satisfiability problems. In Richard J. Lipton, Walter A. Burkhard, Walter J. Savitch, Emily P. Friedman, and Alfred V. Aho, editors, *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA*, pages 216–226. ACM, 1978.

23 Uwe Schöning. A probabilistic algorithm for k-sat and constraint satisfaction problems. In *40th Annual Symposium on Foundations of Computer Science*, pages 410–414, 1999.

24 Rainer Schuler. An algorithm for the satisfiability problem of formulas in conjunctive normal form. *J. Algorithms*, 54(1):40–44, 2005.

25 Mate Soos, Stephan Gocht, and Kuldeep S. Meel. Tinted, detached, and lazy CNF-XOR solving and its applications to counting and sampling. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I*, volume 12224 of *Lecture Notes in Computer Science*, pages 463–484. Springer, 2020.

26 Mate Soos and Kuldeep S. Meel. BIRD: engineering an efficient CNF-XOR SAT solver and its applications to approximate model counting. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 1592–1599. AAAI Press, 2019.

27 Leslie G. Valiant and Vijay V. Vazirani. NP is as easy as detecting unique solutions. *Theor. Comput. Sci.*, 47(3):85–93, 1986.

Twin-Width Is Linear in the Poset Width

Jakub Balabán
Masaryk University, Brno, Czech Republic

Petr Hliněný
Masaryk University, Brno, Czech Republic

Abstract

Twin-width is a new parameter informally measuring how diverse are the neighbourhoods of the graph vertices, and it extends also to other binary relational structures, e.g. to digraphs and posets. It was introduced just very recently, in 2020 by Bonnet, Kim, Thomassé and Watrigant. One of the core results of these authors is that FO model checking on graph classes of bounded twin-width is in FPT. With that result, they also claimed that posets of bounded width have bounded twin-width, thus capturing prior result on FO model checking of posets of bounded width in FPT. However, their translation from poset width to twin-width was indirect and giving only a very loose double-exponential bound. We prove that posets of width d have twin-width at most $8d$ with a direct and elementary argument, and show that this bound is tight up to a constant factor. Specifically, for posets of width 2 we prove that in the worst case their twin-width is also equal 2. These two theoretical results are complemented with straightforward algorithms to construct the respective contraction sequence for a given poset.

2012 ACM Subject Classification Mathematics of computing → Combinatorics; Mathematics of computing → Graph algorithms

Keywords and phrases twin-width, digraph, poset, FO model checking, contraction sequence

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.6

Related Version *Full Version*: `http://arxiv.org/abs/2106.15337`

Funding Supported by the Czech Science Foundation, project no. 20-04567S.

1 Introduction

The new notion of twin-width (of graphs, digraphs, or matrices) was introduced just very recently, in 2020, by Bonnet, Kim, Thomassé and Watrigant [4], and yet has already found many very interesting applications. These applications span from efficient parameterized algorithms and algorithmic metatheorems, through finite model theory, to classical combinatorial questions. See also the series of follow-up papers [2, 1, 3, 5].

We leave formal definitions for the next section. Informally, in simple graphs, twin-width measures how diverse are the neighbourhoods of the graph vertices. E.g., cographs[1] have the lowest possible value of twin-width, 0, which means that the graph can be brought down to a single vertex by successively identifying twin vertices (hence the name, *twin-width*). Two vertices x and y are *twins* if they have the same neighbours in the graph, precisely $N(x) \setminus \{y\} = N(y) \setminus \{x\}$ (the concept of twin-width of graphs does not care about mutual adjacency of the identified vertices).

More generally, imagine we identify arbitrary two vertices x_1, x_2 in a graph G into a new vertex x; then the ordinary neighbours of new x will capture what the former neighbourhoods of x_1 and of x_2 in G have had in common (except each other of x_1, x_2), and we will additionally create new *red edges* from x to those vertices on which the former neighbourhoods of x_1 and

[1] Cographs are the graphs which can be built from singleton vertices by repeated operations of a disjoint union and taking the complement.

of x_2 disagreed in G. Moreover, all other previously created red edges at x_1 or x_2 will stay red and incident to x after the identification. Note that the former vertices x_1, x_2 are removed and no loop is created on x. Precisely, denote by $N(x_i)$ the ordinary ("black") neighbours of x_i in G and by $N_r(x_i)$ the red neighbours of x_i. After the identification of x_1 and x_2 into x, the ordinary neighbours of x will be the vertices of $\big(N(x_1) \cap N(x_2)\big) \setminus \big(\{x_1, x_2\} \cup N_r(x_1) \cup N_r(x_2)\big)$, and the red edges of x will go to the vertices of $\big((N(x_1) \Delta N(x_2)) \cup N_r(x_1) \cup N_r(x_2)\big) \setminus \{x_1, x_2\}$. With respect to this, a graph G has twin-width $\leq d$ if one can reduce G down into a single vertex by successively identifying pairs of its vertices such that the maximum degree of the subgraph on the red edges at every step of this reduction process is at most d.

We are, in particular, interested in the algorithmic metatheorem area. Namely, Bonnet et al. [4] have proved that classes of binary relational structures (such as simple graphs and digraphs) of bounded twin-width have efficient FO model checking algorithms. In one of previous studies on algorithmic metatheorems for dense structures, Gajarský et al. [7] proved that posets (which present a special case of simple digraphs) of bounded width admit efficient FO model checking algorithms. The *width of a poset* is the maximum size of an antichain in it. Since [4] have also proved that posets of bounded width have bounded twin-width, a combination of this finding with the FO model checking algorithm of [4] directly generalizes the algorithmic metatheorem of [7].

The proof of bounded width of posets in [4] is, however, indirect (it uses a characterization by so-called mixed minors of the adjacency matrix) and gives, for posets of width d, only a very loose upper bound of $2^{2^{\mathcal{O}(d)}}$ for the twin-width. Although the proof in [4] is, in principle, constructive, its intricacy makes it really hard to understand why posets of bounded width should have bounded twin-width, and which vertices to identify in the reduction process. In fact, as we will see in this paper, already for posets of width 2 there is no immediate way to optimally choose the pairs for identification.

The main contribution of our paper is in giving direct and tighter constructive linear lower and upper bounds for the twin-width of posets of width d. Precisely, the twin-width of such a poset in the worst case is at least $d-1$ and at most $8d-9$ (Proposition 2.4 and Theorem 3.1). Specifically for posets of width 2, we prove that their twin-width is also at most 2 and this bound cannot be further improved (Theorem 4.1). These results are accompanied by simple and fast algorithms to compute the corresponding contraction sequence.

Our refined results on twin-width of posets provide, in turn, better runtime bounds for FO model checking algorithms of posets [7] and of other classes which have previously been reduced to posets of bounded width, such as [8, 6, 9].

2 Preliminaries and formal definitions

We consider only finite graphs. Our graphs and digraphs are simple, meaning that they do not have parallel edges or loops, except that a simple digraph may have up to one oriented loop per vertex. Formally, a graph is a pair $G = (V, E)$ such that $E \subseteq \binom{V}{2}$, and a digraph is $G = (V, E)$ such that $E \subseteq V \times V$. We deal with (finite) partially ordered sets, shortly *posets*, which we represent as reflexive, antisymmetric and transitive digraphs. Let the *width of a poset* P be the maximum size of an antichain in P, that is the maximum independent set size in the digraph P.

We formally define twin-width using the "matrix-partitioning" view of [4, Section 5], and we restrict ourselves only to the symmetric twin-width which is relevant to graphs and posets. Let $\boldsymbol{A}$ be a square matrix with entries from a finite set L (e.g., $L = \{0, 1\}$ for undirected graphs and $L = \{0, 1, -1, 2\}$ for digraphs, as explained below), and assume that both the

rows and the columns of $\boldsymbol{A}$ are indexed by the same ground set X. Let $\mathcal{R}$ denote any partition of X into nonempty sets. For two parts $R, Q \in \mathcal{R}$, the submatrix of $\boldsymbol{A}$ formed by the rows indexed by R and the columns indexed by Q is called the *$(R \times Q)$ zone* of $\boldsymbol{A}$. Naturally, a zone of $\boldsymbol{A}$ is *constant* if all entries in the zone are equal. For $P \in \mathcal{R}$, the *error value* (the "red degree") of a row part P (column part P) in $\boldsymbol{A}$ is the number of non-constant zones $P \times Q$ (zones $Q \times P$, respectively) in $\boldsymbol{A}$ over all $Q \in \mathcal{R}$ (including $Q = P$).

▶ **Definition 2.1.** *Let $\boldsymbol{A}$ be a square matrix with the rows and columns indexed by a ground set X, $|X| = n$. We say that the* symmetric twin-width of $\boldsymbol{A}$ is at most d *if there exists a sequence of partitions $\mathcal{R}^1, \ldots, \mathcal{R}^n$ of X (a* contraction sequence*) such that;*

- *$\mathcal{R}^1$ is the finest partition of X ($|\mathcal{R}^1| = n$) and $\mathcal{R}^n$ is the coarsest partition of X ($|\mathcal{R}^n| = 1$),*
- *for each $i = 1, \ldots, n-1$, the partition $\mathcal{R}^{i+1}$ results by merging ("contraction" of) some two parts of $\mathcal{R}^i$, and*
- *for each $i = 1, \ldots, n$ and every $P \in \mathcal{R}^i$, the error value of the row P and the column P in $\boldsymbol{A}$ is at most d.*

For a quick illustration, consider Definition 2.1 applied to the adjacency matrix $\boldsymbol{A}_G$ of a graph G. Then the definition of symmetric twin-width of $\boldsymbol{A}_G$ coincides with the twin-width of G as stated in Section 1, except that the diagonal zones ($Q \times Q$ for $Q \in \mathcal{R}^i$ at step i) of $\boldsymbol{A}_G$ are often non-constant, and hence the symmetric twin-width of $\boldsymbol{A}_G$ may be equal or by one higher than the twin-width of G (this difference is neglected in [4]).

For a poset $P = (X, \leq_P)$, viewed as a digraph on the ground set X, we consider the matrix $\boldsymbol{A}_P$ defined as follows (according to [4]); $a_{u,v} = 1$ iff $u \leq_P v$, $a_{u,v} = -1$ iff $v \lneq_P u$, and $a_{u,v} = 0$ otherwise[2]. The *symmetric twin-width of P* is the symmetric twin-width of $\boldsymbol{A}_P$.

For our purpose of giving a closer relation between the width and the twin-width of posets it is, though, much more convenient to use a specialized definition which we call a *natural twin-width* of posets (for a distinction). We shortly write $a \sim_P b$ iff $a \leq_P b$ or $b \leq_P a$ (i.e., the vertices a, b are comparable in P). Our definition reads:

▶ **Definition 2.2** (Natural twin-width of a poset)**.** *A triple $P = (X, \leq_P, R)$ is a* red poset *if $P_0 = (X, \leq_P)$ is a poset and R is a set of unordered pairs of incomparable elements of P_0. The* red degree *of P is the maximum degree of the "red" graph (X, R).*

A contraction *of two vertices $x_1, x_2 \in X$ of P (into a new vertex x) creates the red poset $P' = (X', \leq'_P, R')$ where $X' = (X \setminus \{x_1, x_2\}) \cup \{x\}$ and*

- *$a \leq'_P b$ iff $a \leq_P b$ for all $a, b \in X \setminus \{x_1, x_2\}$, and $x \leq'_P x$,*
- *$a \leq'_P x$ (resp. $x \leq'_P a$) iff $a \leq_P x_1$ and $a \leq_P x_2$ (resp. $x_1 \leq_P a$ and $x_2 \leq_P a$),*
- *$R' = (R \restriction X') \cup R_1 \cup R_2$ where $R_1 = \{\{a, x\} : \{a, x_1\} \in R \vee \{a, x_2\} \in R\}$ and $R_2 = \{\{a, x\} : a \not\sim'_P x \wedge (a \sim_P x_1 \vee a \sim_P x_2)\}$ ($R \restriction X'$ stands for the restriction of R to X').*

In other words, the red edges R' of P' are; (i) those inherited from P which compose of the restriction of R to X' and the red edges R_1 formerly incident to x_1 or x_2 in P, and (ii) the new ones in R_2 between x and those vertices of $X \setminus \{x_1, x_2\}$ which compared in P to x_1 and to x_2 in different ways.

An (ordinary) poset $P_0 = (X, \leq_{P_0})$ has natural twin-width at most d *if the red poset $(X, \leq_{P_0}, \emptyset)$ can be reduced down to a single vertex by a sequence of contractions such that, at each step, the red degree is at most d. So, the natural twin-width of P_0 equals the minimum integer d such that P_0 has natural twin-width at most d.*

[2] If we considered general digraphs (which are not always antisymmetric), we would also consider value $a_{u,v} = 2$ iff both $(u, v), (v, u)$ were edges of the digraph.

With respect to the definition of a contraction (in red posets), the following is a useful convention: If a red poset $P = (X, \leq_P, R)$ resulted by a sequence of contractions from a poset $P_1 = (X_1, \leq_{P_1})$, then a vertex $y \in X$ uniquely corresponds to a set $Y \subseteq X_1$ of those vertices of P_1 which were contracted down to y, and hence we will chiefly refer to Y with the name $y \subseteq X_1$. Consequently, the vertices of such X at the same time form a partition of X_1 in P_1 (and, with negligible abuse of notation, X_1 itself is viewed as the partition of X_1 into singletons) which brings us very close to Definition 2.1 of symmetric twin-width.

In accordance with this convention, we will sometimes denote the vertex resulting from a contraction of the vertices x_1 and x_2 shortly by x_1x_2. See an example in Figure 1.

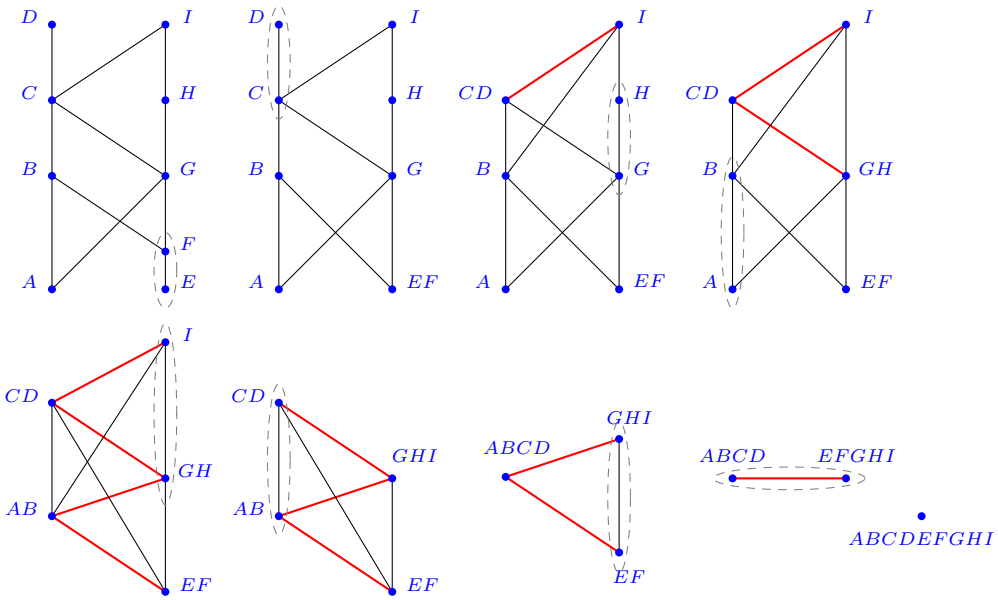


Figure 1 An example of a contraction sequence for the top-left poset (each step contracts the encircled pair), having red degree at most 2. As in Hasse diagrams, the edges (black) of the posets are oriented up, and we skip drawing edges which are implied by reflexivity and transitivity.

▶ **Proposition 2.3.** *If the symmetric twin-width of a poset P is d_s and the natural twin-width of P is d, then $d \leq d_s \leq d + 1$.*

Proof (sketch). We compare Definitions 2.1 and 2.2 for the same contraction sequence; a non-constant zone corresponds to a created red edge, and vice versa. The only difference is at the diagonal zones which may be non-constant while natural twin-width does not consider red loops, and so the symmetric twin-width may be by one higher than the natural one. ◀

2.1 Simple lower bound

▶ **Proposition 2.4.** *There exists a poset of width d and natural twin-width at least $d - 1$.*

Proof. For $d = 2$, we simply take a poset which has no pair of twin vertices, and so any contraction in it creates a red edge, that is, red degree $d - 1 = 1$. E.g., we take the poset formed by the divisibility relation on the set $\{2, 3, 4, 6\}$.

For $d \geq 3$ and $k \geq 4d - 8$, we construct a poset P on a ground set of $n = d(k + 1)$ vertices $C_1 \cup C_2 \cup \ldots \cup C_d$ where each $C_i = \{c_i^0, c_i^1, \ldots, c_i^k\}$ is a chain; $c_i^0 \leq_P c_i^1 \leq_P \ldots \leq_P c_i^k$. In the description of P, we consider indices i "modulo d", formally, we set $c_{d+1}^j = c_1^j$, $\ldots$, $c_{d+d}^j = c_d^j$.

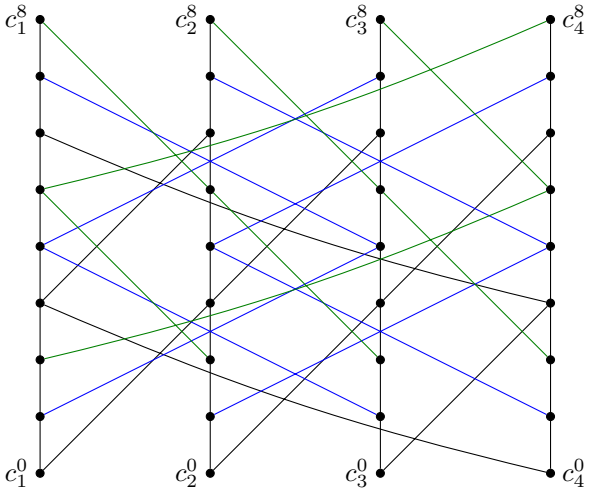


Figure 2 The poset from the proof of Proposition 2.4 for $d = 4$ and $k = 8$, depicted by its Hasse diagram.

Furthermore, for $i = 1, \ldots, d$ and $j = 0, \ldots, k - d + 1$, with $a = 1 + (j \mod (d-1))$, we declare $c_i^j \leq_P c_{i+a}^{j+d-1}$. The rest of $\leq_P$ follows by the reflexive and transitive closure. See an example of this construction in Figure 2, and note that the inverse of P is isomorphic to P (informally, turning P "upside down" gives the same poset) which will be used to reduce the number of cases in the coming arguments by symmetry.

Our aim is to prove that a contraction of any pair of vertices of P already gives red degree $\geq d - 1$. Suppose first that the contracted pair is from the same chain C_i, e.g., c_i^j and c_i^h for $0 \leq j < h \leq k$. We may assume $j \leq k - 2d + 3$, since otherwise we would have $h \geq j + 1 \geq k - 2d + 5 \geq 2d - 3$ and could apply the symmetric argument. Then, for $a = 1 + j \mod (d-1)$, $c_i^j \leq_P c_{i+a}^{j+d-1}$ but $c_i^h \not\leq_P c_{i+a}^{j+2d-3}$. Therefore, the contracted vertex $c_i^j c_i^h$ has at least $(j + 2d - 3) + 1 - (j + d - 1) = d - 1$ incident red edges to the chain C_{i+a}.

Suppose now that the contracted pair is c_i^j and c_y^h where $i \neq y$ and $0 \leq j \leq h \leq k$. Again, we may assume by symmetry that $j \leq k - d + 1$. If $h > j$, then $c_y^h \not\leq_P c_i^{j+d-1}$ and the contracted vertex $c_y^h c_i^j$ has at least $d - 1$ incident red edges to the chain C_i. If $h = j$, then $c_y^h c_i^j$ similarly has at least $2(d-2) \geq d - 1$ incident red edges to the chains C_i and C_y. ◀

3 Upper bound for posets of width d

Complementing Proposition 2.4, we give the core upper estimate followed by its proof:

▶ **Theorem 3.1.** *A poset of width $d \geq 2$ has natural twin-width at most $8d - 9$, and hence the symmetric (matrix) twin-width at most $8d - 8$. The corresponding contraction sequence can be found in time $\mathcal{O}(dn^2)$ where n is the number of vertices of the poset.*

By Dilworth's theorem, a poset $P = (X, \leq_P)$ is of width d if and only if the ground set X can be partitioned into at most d *chains* (a chain is linearly ordered by $\leq_P$). Hence, from now on, we will consider a poset of width d with a fixed partition π of X into d (nonempty) chains, formally as a triple $P = (X, \leq_P, \pi)$ where $\pi = \{U_1, \ldots, U_d\}$.

Our upper bound in Theorem 3.1 will use only a special type of contractions – of two consecutive vertices of the same chain of P. Since contractions inside a chain essentially preserve the chain partition of P, we will for simplicity refer to the new chain partition as to

π again. We shall thus work with the following special kind of red posets, which result from a chain-partitioned poset by our special contractions:

▶ **Definition 3.2** (red d-neighbourly poset). *Let $P_0 = (X_0, \leq_{P_0}, \pi)$ be a poset partitioned by π into d chains. A* neighbourly contraction *is a contraction of a vertex pair x_1, x_2 such that $x_1 \neq x_2$ belong to the same chain U of π and they are consecutive in this chain (i.e., no element of U is strictly between x_1 and x_2). Note, however, that for such a pair x_1, x_2 there could exist a vertex y in another chain of π such that $x_1 \lneq y \lneq x_2$.*

A tuple $P = (X, \leq_P, \pi, R)$ is called a red d-neighbourly poset *(shortly a* neighbourly poset*) if the red poset $(X, \leq_P, R)$ is obtained from P_0 by an arbitrary sequence of neighbourly contractions (we shortly say that P is a* contraction of P_0*).*

Roughly speaking, our proof of Theorem 3.1 is going to argue that, although some neighbourly contractions can create many new red edges, overall the number of red edges that can potentially be created by every neighbourly contraction is only proportional to the size of the poset. Therefore, one can always find a "good" neighbourly contraction. Later on in the contraction sequence, we also have to watch the number of previously created red edges which is straightforward. Of course, to fulfill Definition 2.2, we will have to eventually contract the remaining d single-vertex chains into one, but that part will be a trivial conclusion at the end of our proof.

When dealing with neighbourly posets such as $P = (X, \leq_P, \pi, R)$, we adopt some special notation. Consider a chain $U = (u_1, u_2, \ldots, u_m)$ ordered as $u_1 \leq_P \ldots, \leq_P u_m$. For a vertex $x \in X$ such that $x = u_a$ of U, we shall write x^{+i} for the vertex u_{a+i} and x^{-i} for the vertex u_{a-i} (of course, assuming $a+i \leq m$ or $a-i \geq 1$, respectively). Let x^+ and x^- be a shorthand for x^{+1} and x^{-1}.

We give a unified way of picturing neighbourly posets – a *chain diagram* (already seen in Figure 1), which is close to the traditional Hasse diagram of a poset, but not exactly the same.

▶ **Definition 3.3** (chain diagram). *Let P be a red d-neighbourly poset. Every chain of P is drawn as a vertical line, the red edges of R are drawn as* red bars *between pairs of the chains, and there is a* black bar *from a vertex u of a chain U to a vertex v of a chain $V \neq U$, if and only if v is the least vertex of V greater than u and u is the greatest vertex of U smaller than v.[3] A black bar is never drawn as horizontal and is implicitly directed up in the picture.*

3.1 Structure of black and red bars

While black bars of a neighbourly poset P are directed by $\leq_P$, that is we have a black bar (u, v) if $u \leq_P v$ as in Definition 3.3, red bars are by Definition 2.2 undirected. Nevertheless, we can assign a direction to a red bar $\{u, v\} \in R$ as follows.

▶ **Definition 3.4** (orienting the red bars). *Let the neighbourly poset $P = (X, \leq_P, \pi, R)$ be a contraction of an ordinary poset $P_0 = (X_0, \leq_{P_0}, \pi)$, and recall that we view $u \in X$ as the subset $u \subseteq X_0$ of the respective contracted vertices of P_0. Since $u \subseteq X_0$ belongs to one chain of P_0 by Definition 3.2, the minimum $min(u)$ of u is well-defined.*

We orient the red bar $\{u, v\} \in R$ as (u, v), from u to v, if $min(u) \leq_{P_0} x$ for some $x \in v$, but $min(v) \not\leq_{P_0} min(u)$. Though, if both directions (u, v) and (v, u) are assigned by this criterion (which is possible, e.g., when the minima of u and v are incomparable), then we choose (u, v) if the last contraction into u happened later than that into v.[4]

[3] Notice that since R contains only incomparable pairs, there cannot be a red and a black bar together between the same pair.

[4] The latter criterion of choosing between (u, v) and (v, u) is not really important; we introduce it only to "break the tie" in a deterministic way.

Observe that Definition 3.4 assigns exactly one orientation to each $\{u,v\} \in R$. As an informal explanation, a red bar $\{u,v\} \in R$ in P means that between the sets $u, v \subseteq X_0$ in P_0, the edges (and non-edges) are not uniform (not all in one direction), and then we choose the "prevailing direction" for the orientation of $\{u,v\}$. We shall write a red bar as $\{u,v\} \in R$ if we do not care about the orientation of it, and as $(u,v) \in R$ if we do care.

Now we summarize some (basically trivial) technical properties used in further proofs.

▶ **Lemma 3.5.** *Let $P = (X, \leq_P, \pi, R)$ be a d-neighbourly poset and U and V be two distinct chains of P determined by π.*

a) *If $u \leq_P v$ where $u \in U$ and $v \in V$, then there is no $i > 0$ such that $(u, v^{+i}) \in R$. Analogously, if $u \geq_P v$, then there is no $i > 0$ such that $(v^{-i}, u) \in R$.*

b) *If $(u,v) \in R$ where $u \in U$ and $v \in V$ such that $(u, v^+) \notin R$, then $u \leq_P v^+$ (informally, red bars oriented from u to vertices of V come in a consecutive strip "capped" by a black bar). An analogous claim symmetrically holds for red bars oriented towards u from V.*

c) *If $(u,v) \in R$ where $u \in U$ and $v \in V$, then there are no $i, j > 0$ such that $(u^{+i}, v^{-j}) \in R$ or $u^{+i} \leq_P v^{-j}$ (informally, no two red bars of the same orientation from U to V may "cross", and no black bar from U to V may be "crossed" by a red bar starting below it in U).*

d) *There are together at most $|U| + |V| - 1$ red bars from a vertex of U to a vertex of V.*

e) *Assume $u \in U$, $v \in V$ such that $(u,v) \in R$ is a red bar that has been created by a neighbourly contraction into u, and that no contraction into v has happened after the creation of red (u,v). Then no neighbourly contraction in P in the chain U can create another red bar oriented towards v.*

An analogous claim holds for $(v,u) \in R$ and creation of red bars oriented from v.

Proof. Let P be a contraction of the ordinary poset $P_0 = (X_0, \leq_{P_0}, \pi)$.

a) Trivially by transitivity, $u \leq_P v \leq_P v^{+i}$ contradicts that pairs in R are incomparable.

b) By $(u,v) \in R$ and Definition 3.4, for some $x \in v$ of P_0 we have $min(u) \leq_{P_0} x \leq_{P_0} min(v^+)$. Then, if $max(u) \not\leq_{P_0} min(v^+)$, we would have forbidden $(u, v^+) \in R$. Therefore, by homogeneity of the edges from u to v^+ in P_0, we get desired $u \leq_P v^+$.

c) Assume the contrary, that $(u^{+i}, v^{-j}) \in R$. Then, by Definition 3.4 and transitivity in P_0, $max(u) \leq_{P_0} min(u^{+i}) \leq_{P_0} x \leq_{P_0} min(v)$ where $x \in v^{-j}$, which contradicts the assumption $\{u,v\} \in R$. The same argument goes through if $u^{+i} \leq_P v^{-j}$ with $x = min(v^{-j})$.

d) We ignore the chains other than U or V, and we prove the statement by induction. The base case is $|U| = 1$; clearly, there may be at most $|V| = |U| + |V| - 1$ red bars going from U to V. Now, let u_1 be the minimum of the chain U, and assume that there are $q \geq 0$ red bars oriented from u_1 to V. Let $V_1 \subseteq V$ be the lowest $q-1$ vertices of the chain V. By (c), only red bars from u_1 may end in V_1. So, we remove the vertices $\{u_1\} \cup V_1$ and, by induction, there are at most $|U| - 1 + |V| - (q-1) - 1 = |U| + |V| - q - 1$ red bars from $U \setminus \{u_1\}$ to $V \setminus V_1$. With the q red bars starting in u_1, we get the desired bound.

e) If $(u,v) \in R$ has been created by a contraction into u, and not by a prior contraction into v, then $min(u) \leq_{P_0} min(v)$ using Definition 2.2. Hence a further neighbourly contraction in the chain U below u cannot at all create a red bar incident to v, and a neighbourly contraction in the chain U above u can only create a new red bar oriented from v, according to Definition 3.4. ◀

3.2 Minimizing the red potential

Now comes the core of the proof of Theorem 3.1, estimating how many red edges can potentially result from all possible neighbourly contractions in P. Let $u \in U$ be a vertex of a chain U which is not maximal, u^0 be the vertex created by the contraction of u and u^+, and

define the *red potential of* u as the number of red edges incident to u^0 after the contraction (so, previous red edges incident to u or u^+ are also counted here). The *red potential of the chain* U is simply the sum of red potentials over the non-max vertices of U, and the *red potential of* P is the sum over all chains of P.

▶ **Lemma 3.6.**

a) *If $P = (X, \leq_P, \pi, R)$ is a red d-neighbourly poset with $m = |X|$ elements, then the red potential of P is at most $2(d-1)(m-d) + 2|R|$.*

b) *There are at most $|R| \leq 2(d-1)(m-d)$ red bars in P.*

Proof. In both parts of this proof, let U and V be any two distinct chains of P.

a) For any vertex $x \in U$ (resp. $x \in V$), let the *red supplement of* x, denoted $rs(x)$, be the number of neighbourly contractions in V (resp. in U) that create a red bar incident to x in the ordinary poset $P_0 = (X, \leq_P, \pi)$, i.e., if we ignore the red bars of P. Let $\rho_{U,V}$ be the sum of $rs(x)$ over all $x \in U \cup V$. Notice that $\rho_{U,V}$ is the red potential of P_0 restricted to $U \cup V$.

By Definition 3.4, $rs(x) \leq 2$, since at most one neighbourly contraction creates a new red bar oriented to x and at most one creates a new red bar oriented from x, and hence $\rho_{U,V} \leq 2(|U| + |V|)$. We slightly improve this immediate bound as follows. Let u_1 and v_1 be the minima of the chains U, V, and let u' and v' be their maxima. If $rs(u_1) + rs(v_1) > 2$ then, up to symmetry, $rs(u_1) = 2$. This would mean that there is a vertex $v \in V$ such that $v \leq u_1$. Since $v_1 \leq v$, we get $v_1 \leq u_1 \leq u$ for all $u \in U$, which implies that $rs(v_1) = 0$. Consequently, $rs(u_1) + rs(v_1) \leq 2$, and $rs(u') + rs(v') \leq 2$ by symmetry. Summing with the remaining vertices of $U \cup V$ we obtain an estimate $\rho_{U,V} \leq 2(|U| + |V|) - 4$.

Let the cardinalities of the d chains of P be $m_1, m_2, \ldots, m_d$. Summing the latter estimate over all $\binom{d}{2}$ unordered pairs of chains of P, we get

$$\sum_{1 \leq i < j \leq d} \rho_{U_i,U_j} = \sum_{1 \leq i < j \leq d} \big[\, 2 \cdot (m_i + m_j) - 4 \,\big] = 2(d-1)m - 4 \cdot \binom{d}{2} = 2(d-1)(m-d)\,. \quad (1)$$

Now it remains to add the contribution of the red bars of R in P. Each red bar $(u, v) \in R$, where $u \in U$, contributes to the red potentials of at most two neighbourly contractions in U; namely to those of the pairs u^-, u and u, u^+. However, we are over-counting this way, and we now show that it is enough to count a "+1" contribution towards one of the two contractions. Precisely, we contribute a red bar (u, v) to the contraction of the pair u, u^+ since the following holds: If $(u^-, v) \in R$, then the contraction of u^-, u into u^0 anyway makes only one inherited red bar (u^0, v) (out of the two red bars (u, v) and (u^-, v) of P) and our rule contributes (u^-, v) to the pair u^-, u. Otherwise, by Lemma 3.5(b), we have $u^- \leq_P v$ and the inherited red bar (u^0, v) has already been counted as a new one in $rs(v)$ above.

Since each red bar in R increases the red potential in two chains, we in total get that the red potential of P is at most $2(d-1)(m-d) + 2|R|$.

b) A straightforward application of Lemma 3.5(d) gives an upper bound of $2(|U| + |V|) - 2$ on the number of red bars between chains U and V (in both directions). However, if u_1, v_1 denote the minima of U, V, then $(u_1, v_1) \notin R$ up to symmetry. So, an application of Lemma 3.5(d) to $U \setminus \{u_1\}$ and V gives at most $|U| + |V| - 2$ red bars from U to V. A symmetric "saving" can be obtained for the maxima of U, V, and hence we can have at most $2(|U| + |V|) - 4$ red bars between chains U and V in both directions.

Finally, summing the latter bound over all pairs of chains as in (1), we get the desired bound $|R| \leq 2(d-1)(m-d)$. ◀

Proof of Theorem 3.1. Let $P_0 = (X_0, \leq_{P_0}, \pi)$ be an ordinary poset partitioned into d chains. We are now ready to finish the main proof; to find a desired contraction sequence of P_0 of bounded red degree. The natural idea at each step (with a neighbourly poset P obtained from P_0 so far) is to exhaustively find a neighbourly contraction in P of the smallest red potential, which is upper-bounded independently of the size of P based on Lemma 3.6.

Let $P = (X, \leq_P, \pi, R)$ be a contraction of P_0, as above, and $m = |X| \leq |X_0| = n$. The red potential of whole P is at most $2(d-1)(m-d) + 4(d-1)(m-d) = 6(d-1)(m-d)$ by Lemma 3.6. Since there are $m - d$ possible neighbourly contractions in P, one of them has the red potential at most $6(d-1)$. This of course makes sense only for $m > d$ but for $m \leq d$, we can finish the contraction sequence arbitrarily, without getting a high red degree.

We are nearly done, but there is a small catch. For a vertex $x \in X$ of P, call a red edge $\{x, y\}$ incident to x *domestic* (to x) if it has been there already the last time we have contracted into x along our contraction sequence; otherwise, call red $\{x, y\}$ *foreign*. While the argument in the previous paragraph bounded the number of domestic red edges incident to any vertex along the whole sequence, we have not yet bounded the number of potential foreign red edges (that is those which have been created by contraction to other vertices later on). Using Lemma 3.5(e), we claim that there can be at most one foreign edge incident to x oriented towards x, and one oriented from x, per each other chain of P. Hence the number of foreign red edges incident to any x is at most $2(d-1)$, and at most $2d-3$ if x has also some domestic red edges (again by Lemma 3.5(e)). Altogether, the maximum red degree along our contraction sequence is at most $6(d-1) + 2d - 3 = 8d - 9$.

The above proof straightforwardly translates into a simple and efficient algorithm. The red potential of one vertex of P can be found in time proportional to d and the value of this red potential, and hence the minimum red potential of the poset P in the current step of a contraction sequence is determined in time $\mathcal{O}(dm) \leq \mathcal{O}(dn)$. The same time is sufficient to update P for the next step. Since we need $n - 1$ steps of the contraction sequence for P_0, this computation is finished in time $\mathcal{O}(dn^2)$. ◀

4 Tight estimate for posets of width 2

From Section 3 we get that in the worst case a poset of width 2 has twin-width at least 1 and at most 7 (where the upper bound of Theorem 3.1 can likely be improved a bit in this special case of $d = 2$). However, in order to get an exact worst-case value of twin-width, namely value 2, we had to employ a very different approach – one which is special only for posets of width 2 (unfortunately, this argument does not generalize even to width 3).

We start with the upper estimate:

▶ **Theorem 4.1.** *A poset of width 2 has natural twin-width at most 2, and the corresponding contraction sequence can be computed in linear time.*

We prove the statement by providing the claimed algorithm and proving its correctness. On a high level, our algorithm performs a depth-first search for a "safe" possibility of a neighbourly contraction in one of the two chains of a poset P, starting from a minimal vertex u_1 of the poset. By a safe neighbourly contraction we mean one in which we have or create at most one incoming and at most one outgoing red bar in the contracted vertex. We also stay in firm control of all red bars in intermediate contracted red posets.

To control the search for neighbourly contraction pairs and the occurrence of red bars, we introduce the notion of a *directed bar path* (recall also Definition 3.3 of a chain diagram with bars). A bar path in a red 2-neighbourly poset P is a directed path represented as a vertex sequence $(x_1, x_2, \ldots, x_k)$ in P such that

- x_1 is a minimal vertex of P (either of the possible two),
- for $i = 1, \dots, k-1$, (x_i, x_{i+1}) is a black or red bar in P oriented this way, and
- if both x_i^+, x_{i+1}^+ exist in P, then $x_i^+ \not\leq_P x_{i+1}^+$.

Notice that a bar path is "zig-zag" switching between the two chains of P. Our bar-path controlled algorithm is then formalized in Algorithm 1.

Commenting on this algorithm, we remark that the main part (the one searching for a "safe" neighbourly contraction along bar path B) is presented on lines 15 to 26. The preceding supplementary part on lines 9 to 13 is there to prepare for a possible contraction at the root u_1 of bar path B; if u_1 is not the global minimum of P, then the lower vertices of the other chain V of P could create many red bars oriented towards u_1 after neighbourly contraction to u_1. This is eliminated by safe contractions of the problematic vertices of V to v_1 on line 13, which make future neighbourly contractions to u_1 safe as well.

Actually, the course of Algorithm 1 is illustrated in previous Figure 1. It is $u_1 = A$, $v_1 = E$ (after the supplementary first step, $v_1 = EF$), and the bar path leading to the first contraction (of C, D) on line 22 is $B = (A, G, C)$. Further on, for example, in the fourth picture (the top-right red poset) we have $B = (A)$ and $B^+ = (A, GH, CD, I)$. In the fifth picture (bottom-left), we get $B = (AB, GH)$ and $B^+ = (EF, AB, GH, CD, I)$, and so on.

Proof of Theorem 4.1. We refer to Algorithm 1 and the definition of bar path B. Let P_0 be the input ordinary poset and P the current 2-neighbourly poset, as in the algorithm. Let U_0, V_0 denote the two chains of P_0 and $u_1 = min(U_0)$ be the start (root) of bar path B which stays fixed during the course of computation. Let $v_1 = min(V_0)$. Note that we slightly abuse notation by referring to these vertices as to u_1 and v_1 also in the poset P, after possible contractions into u_1 or v_1.

For the purpose of analysis of Algorithm 1, we define an *extended bar path* $B^+ \supseteq B$ of the current bar path B in P as follows: B^+ starts with (v_1, u_1) if $(v_1, u_1) \in R$, and B^+ starts in u_1 otherwise. Then B^+ contains all bars of B in order, then possibly one black bar starting in the last vertex of B, and finally, B^+ ends as a directed path using only red bars of P. We claim the following *invariant* at the beginning and after every iteration of the loop from line 6:

(I) B conforms to the conditions of a bar path.

(II) There exists an extended bar path B^+ of B in P containing all red bars of P.

To better understand the role of an extended bar path, observe that, modulo renaming of contracted vertices, B^+ coincides with the former bar path in the iteration in which the upper-most red bar of B^+ has been created by a contraction.

Since an extended bar path is acyclic and does not repeat vertices – this is not trivial but follows from Definition 3.4 – condition (II) then implies that the red degree of P after every iteration is at most 2, thus proving the conclusion of Theorem 4.1. Our aim hence is to prove the invariant, by induction on the iterations of the main cycle.

At the beginning, $B = (u_1)$ satisfies (I) and (II) is trivial since $R = \emptyset$. We now assume that these hold when an iteration of the main cycle (line 6) starts. Possible contractions on line 13 do not create red edges or change B, except that when $|U| = 1$ they could create the red bar (v_1, u_1) which will be included in our extended bar path there.

For the next arguments, note that line 15 always selects a red bar (u, v) if there is one starting from u. So, if there is no red bar from u and $u^+ \leq_P v$, then (u, v) is *not* a black bar either (Definition 3.3) and the extended bar path B^+ from (II) before this iteration cannot reach v. Consequently, the contraction of u, u^+ on line 22 of this iteration does not make

Algorithm 1 Constructing a contraction sequence of red degree 2.

Input: Given a poset $P_0 = (X_0, \leq_{P_0}, \pi)$ partitioned into 2 chains.
Output: A record of the constructed contraction sequence of P_0 of red degree ≤ 2.

1: **declare** $P = (X, \leq_P, \pi, R)$ a red 2-neighbourly poset
2: **declare** B a directed bar path (of black and red bars of P)
3: $P \leftarrow$ red neighbourly poset $P = (X_0, \leq_{P_0}, \pi, \emptyset)$;
4: $B \leftarrow$ bar path (u_1), where u_1 is a minimal vertex of P_0 (any of possible two);
5: *// bar path B directs the course of the algorithm (kind of similarly to classical DFS); note that B stays rooted in $u_1 = min(U)$ till the end of the main loop*
6: **while** $B \neq \emptyset$ **do**
7: $u \leftarrow$ the (upper) end vertex of bar path B, i.e., $B = (x_1, \ldots, x_k, u)$;
8: $\{U, V\} \leftarrow$ the two chains of P (by π) such that $u \in U$;
9: **if** $B = (u)$, which is equivalent to $u = min(U)$ **then**
10: *// here we contract the lower section of V which is homogeneous towards U:*
11: $v_1 \leftarrow min(V)$; $u_2 \leftarrow$ smallest $u_2 \in U$ such that $v_1 \leq_P u_2$, or u_2 nonexistent;
12: **while** $|V| > 1$ and $\big(|U| = 1$ or (u_2 exists and $v_1^+ \leq_P u_2)\big)$ **do**
13: $P \leftarrow$ **contraction** of the pair v_1, v_1^+ in P; $v_1 \leftarrow v_1 v_1^+$;
14: *// the main part; prolonging B, or possibly contracting at the end and shortening B:*
15: $v \leftarrow$ the smallest vertex $v \in V$ such that $u \leq_P v$ or $(u, v) \in R$, or v nonexistent;
16: **if** v, v^+ and u^+ exist in P, and $u^+ \not\leq_P v^+$ **then**
17: *// note that since $u^+ \not\leq_P v^+$, we get that $(u, v) \in R$ or (u, v) is a black bar*
18: $B \leftarrow (B, v)$, i.e., prolongation of bar path B by the bar (u, v);
19: **else**
20: **if** u^+ exists in P **then**
21: *// the contraction here is safe; even if v, v^+ exist, we have $u^+ \leq_P v^+$ and no red bar towards v^+ (or further up) is created*
22: $P \leftarrow$ **contraction** of the pair u, u^+ in P; $u \leftarrow uu^+$;
23: **if** $B = (u)$ **then**
24: **if** u^+ nonexistent in P **then** $B \leftarrow \emptyset$;
25: **else**
26: $B \leftarrow B \setminus (u)$, i.e., shortening of bar path B by the last bar towards u;
27: $P \leftarrow$ **contraction** of the remaining pair $min(U), min(V)$ in P;

(u, v) red, and (I) and (II) are satisfied after the iteration with the same B^+. Hence we can assume that $(u, v) \in R$ or $u^+ \not\leq_P v$ hold in the coming case analysis.

We now analyze the remaining cases according to the "if" statements from line 16 onward:

- If v, v^+ and u^+ exist and $u^+ \not\leq_P v^+$, then (u, v) is a red or black bar in P, and so we explicitly satisfy all three conditions of a bar path for the prolongation (B, v) on line 18. No new red bars are created in this iteration, and we claim that an extended bar path B^+ from the previous iteration contains or will contain the new bar (u, v); this is trivial if (u, v) is red, and for a black bar (u, v) it is the only bar of P starting in u anyway.
- If u^+ and v exist, and v^+ is nonexistent or $u^+ \leq_P v^+$, then the contraction of u, u^+ on line 22 makes (u, v) red if it was not such before. No other red bar exists or is created from u (although, the bar of B towards u may already be red). Altogether, we inherit

B^+ from the previous step, and with (now) red (u, v) this will be (II) a valid extended bar path of the shortened bar path $B \setminus (u)$ at the end of the iteration. Not to forget (I), $B \setminus (u)$ will be a valid bar path as well.

- If $B = (u)$ in the previous case, we do not shorten B (since u^+ exists), but the arguments stay the same.
- if u^+ exists but v does not, then an extended bar path cannot reach beyond u. So, after the contraction and shortening B by u we will trivially satisfy (I) and (II) again.
- Lastly, if u^+ is nonexistent, then we only shorten B by u, or we are at the end of the algorithm if u is the root of B (note that here the procedure on lines 9 to 13 also takes part and shortens V to one vertex before we stop the cycle from line 6).

It remains to argue why the algorithm stops, and what is the runtime. The first part is clear since every iteration of the main loop either prolongs the current bar path (which is bounded), or eventually finds a next contraction pair.

As for the runtime, we use the following special representation of the working poset P, which extends the chain diagram of Definition 3.3: For every $u \in U$ we record, besides the red bars of u, the least $v \in V$ such that $u \leq_P v$. We input the poset P_0 as a traditional Hasse diagram, find its chain cover by standard means and prepare our special representation of it in linear time with respect to $|X_0|$. The total number of iterations of the main cycle is linearly proportional to the number of performed contractions in the main part. Then, at every iteration of the main cycle, we can perform the computation, and the update of the structure representing P, in constant time. The only exception is a possible iteration of the inner cycle on line 12, which is counted amortized towards the total number of contractions. ◀

The lower estimate matching Theorem 4.1 is as follows.

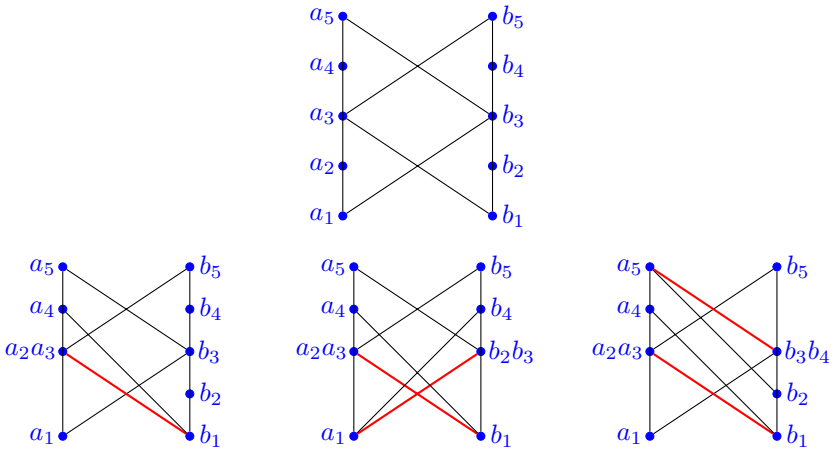


Figure 3 [top] A poset of width 2 which has natural twin-width (at least) 2. [bottom] All possible contractions of this poset, up to symmetry, which have red degree 1.

▶ **Proposition 4.2.** *The poset depicted in Figure 3 has natural twin-width at least* 2.

Proof. [5] Thanks to symmetries in the depicted poset P, it is routine and easy to verify that every contraction of a pair in P results in two red edges incident to the contracted vertex, except the contraction of the pair a_2, a_3 (or the symmetric pairs b_2, b_3 or b_3, b_4 or a_3, a_4). The result of this contraction, a red poset P_1, is on bottom left of Figure 3.

[5] To be safe, we have independently verified this whole proof by an exhaustive computer check.

Now, in P_1, the contraction of (now red) pair a_2a_3, b_1 creates three incident red edges. For every other contracted pair in P_1, we either get the same two red edges as if it was contracted in P, or another red edge incident to a_2a_3 or to b_1, or one of the further two possibilities of isolated red edges depicted at the bottom of Figure 3. In those cases, again by a boring but routinely easy case analysis, one can see that every contraction creates red degree at least two. ◀

5 Conclusions

We have proved a tight relation up to a multiplicative constant between the width of a poset and its twin-width. The constants in this linear relation are very reasonable, but they can likely still be improved. The result also gives a simple and practically usable approximation algorithm for the twin-width of posets of small width, while the same is not known in the more general cases of all graphs or all posets.

References

1 Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width III: max independent set and coloring. *CoRR*, abs/2007.14161, 2020. arXiv:2007.14161.

2 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.

3 Édouard Bonnet, Ugo Giocanti, Patrice Ossona de Mendez, and Stéphan Thomassé. Twin-width IV: low complexity matrices. *CoRR*, abs/2102.03117, 2021. arXiv:2102.03117.

4 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 601–612. IEEE, 2020. doi:10.1109/FOCS46700.2020.00062.

5 Édouard Bonnet, Jaroslav Nešetřil, Patrice Ossona de Mendez, Sebastian Siebertz, and Stéphan Thomassé. Twin-width and permutations. *CoRR*, abs/2102.06880, 2021. arXiv:2102.06880.

6 Simone Bova, Robert Ganian, and Stefan Szeider. Model checking existential logic on partially ordered sets. *ACM Trans. Comput. Log.*, 17(2):10:1–10:35, 2016. doi:10.1145/2814937.

7 Jakub Gajarský, Petr Hliněný, Daniel Lokshtanov, Jan Obdržálek, Sebastian Ordyniak, M. S. Ramanujan, and Saket Saurabh. FO model checking on posets of bounded width. In *FOCS*, pages 963–974. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.63.

8 Robert Ganian, Petr Hliněný, Daniel Král, Jan Obdržálek, Jarett Schwartz, and Jakub Teska. FO model checking of interval graphs. *Log. Methods Comput. Sci.*, 11(4), 2015. doi:10.2168/LMCS-11(4:11)2015.

9 Petr Hliněný, Filip Pokrývka, and Bodhayan Roy. FO model checking on geometric graphs. *Comput. Geom.*, 78:1–19, 2019. doi:10.1016/j.comgeo.2018.10.001.

Dynamic Kernels for Hitting Sets and Set Packing

Max Bannach
Universität zu Lübeck, Germany

Zacharias Heinrich
Universität zu Lübeck, Germany

Rüdiger Reischuk
Universität zu Lübeck, Germany

Till Tantau
Universität zu Lübeck, Germany

Abstract

Computing small kernels for the hitting set problem is a well-studied computational problem where we are given a hypergraph with n vertices and m hyperedges, each of size d for some small constant d, and a parameter k. The task is to compute a new hypergraph, called a *kernel*, whose size is polynomial with respect to the parameter k and which has a size-k hitting set if, and only if, the original hypergraph has one. State-of-the-art algorithms compute kernels of size k^d (which is a polynomial kernel size as d is a constant), and they do so in time $m \cdot 2^d \operatorname{poly}(d)$ for a small polynomial $\operatorname{poly}(d)$ (which is a linear runtime as d is again a constant).

We generalize this task to the *dynamic* setting where hyperedges may continuously be added or deleted and one constantly has to keep track of a size-k^d hitting set kernel in memory (including moments when no size-k hitting set exists). This paper presents a *deterministic* solution with *worst-case* time $3^d \operatorname{poly}(d)$ for updating the kernel upon hyperedge inserts and time $5^d \operatorname{poly}(d)$ for updates upon deletions. These bounds nearly match the time $2^d \operatorname{poly}(d)$ needed by the best static algorithm per hyperedge. Let us stress that for constant d our algorithm maintains a dynamic hitting set kernel with *constant, deterministic, worst-case* update time that is independent of n, m, and the parameter k. As a consequence, we also get a deterministic dynamic algorithm for keeping track of size-k hitting sets in d-hypergraphs with update times $O(1)$ and query times $O(c^k)$ where $c = d - 1 + O(1/d)$ equals the best base known for the static setting.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Fixed parameter tractability

Keywords and phrases Kernelization, Dynamic Algorithms, Hitting Set, Set Packings

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.7

Related Version *Technical Report*: `https://eccc.weizmann.ac.il/report/2019/146/` [3]

1 Introduction

The hitting set problem is a fundamental combinatorial problem that asks, given a hypergraph, whether there is a small vertex subset that intersects ("hits") each hyperedge. Many interesting problems reduce to it: a dominating set of a graph is just a hitting set in the hypergraph that contains for every vertex a hyperedge consisting of the vertex's closed neighborhood; for any fixed graph H, the question of whether we can delete k vertices from a graph G in order to make G an H-free graph can be reduced to the hitting set problem for the hypergraph to which each occurrence of H in G contributes one hyperedge – and this problem in turn generalizes problems such as TRIANGLE-DELETION and CLUSTER-VERTEX-DELETION [1]. The hitting set problem also finds applications in the area of descriptive complexity, as a fragment of first-order logic can be reduced to it [9].

The hitting set problem is NP-complete [25] and its parameterized version p_k-HITTING-SET is W[2]-complete [14]. However, if we restrict the size of hyperedges to at most some constant d, the resulting problem p_k-d-HITTING-SET lies in FPT [20] and even has polynomial kernels. In particular, $d = 2$ is the vertex cover problem, which is still NP-complete, but one of the best-investigated parameterized problems. Already the jump from $d = 2$ to $d = 3$ turns out to be nontrivial in this setting. In detail, the inputs for our algorithms are a hypergraph $H = (V, E)$ and an upper bound k for the size of a hitting set X wanted (a set for which $e \cap X \neq \emptyset$ holds for all $e \in E$). We think of the numbers $n = |V|$ and $m = |E|$ as large numbers, of k as a (relatively small) parameter, and of $d = \max_{e \in E} |e|$ as a small constant (already the cases $d = 3$ and $d = 4$ are of high interest).

Parameterized algorithms for the hitting set problem proceed in two steps: First, the input (H, k) is *kernelized,* which means that we quickly compute a (small) new hypergraph K such that H has a size-k hitting set iff K has one. Afterwards the problem is solved on K using an expensive algorithm based on search trees or iterative compression. The currently best algorithm for computing a kernel is due to Fafianie and Kratsch [17], see also [4, 32] for some recent developments. The cited algorithm outputs a kernel of size k^d (meaning that K has at most k^d hyperedges) in time $m \cdot 2^d \operatorname{poly}(d)$ (meaning that time $2^d \operatorname{poly}(d)$ is needed on average per hyperedge of H). The best algorithms for solving the hitting set problem on the computed kernel K run in time $O(c^k)$, where the exact value of $c = d - 1 + O(1/d)$ is a subject of ongoing research [33, Section 6], [19, 21, 28], and [10, Section 4]. In summary, on input (H, k) one can solve the hitting set problem in time $O(2^d \operatorname{poly}(d) \cdot m + c^k)$.

Our objective in this paper is to transfer (only) the first part of solving the hitting set problem (namely the computation of the kernel K) into the dynamic setting. Instead of a single hypergraph H being given at the beginning, there is a sequence $H_0, H_1, H_2, H_3, \ldots$ of hypergraphs each of which differs from the previous one by a single edge being either added or deleted. One continuously has to keep track of hitting set kernels $K_0, K_1, K_2, K_3, \ldots$ for the current H_i (including moments when H_i has no size-k hitting set). Our aim is to compute the updated kernel K_{i+1} from K_i in constant time based solely on the knowledge which edge was added to or deleted from H_i in order to obtain H_{i+1}.

Doing the necessary bookkeeping to dynamically manage a hitting set kernel is not easy. As an example, consider two hypergraphs H and $\tilde{H}$ with disjoint vertex sets, were H is a clear no-instance (like a matching of size $k + 1$) while $\tilde{H}$ is a hard, borderline case that can only be reduced to a relatively large kernel $\tilde{K}$. A dynamic kernel algorithm that works on $H \dot{\cup} \tilde{H}$ must be able to cope with the situation that we first add all the edges of H (at which point a natural kernel would be a trivial size-1 no-instance K), followed by all the edges of $\tilde{H}$ (which even reinforces that the trivial no-instance K is a correct kernel for the ever-larger hypergraph), followed by a deletion of the edges from H. At some point during these deletions, a dynamic kernel algorithm must switch from the constant-size K to the large kernel $\tilde{K}$. Previous work from the literature [2] shows that it is already tricky to achieve this switch in time polynomial in the size of kernels K and $\tilde{K}$. The challenge we address is to do the updates in *constant worst-case* time, which forces our dynamic algorithm to spread the necessary changes over time while neither resorting to amortization nor to randomization.

Note that we only give a dynamic algorithm for keeping the *kernel* up-to-date with constant update times – we make no claims concerning the time needed to actually compute a hitting set for the current kernel K_i (and, thus, for the current H_i). Phrased in terms of dynamic complexity theory, there are two different problems for which we present algorithms with differing *update times* (the time needed for updating internal data structures) and *query times* (the time needed to construct an output upon request): For the first problem of (just)

computing hitting set kernels K for inputs H, we present a dynamic algorithm with *constant* update time and *zero* query time (since the current kernel K_i is explicitly stored in memory as an adjacency matrix at all times). For the second problem of computing size-k hitting sets X for inputs H, our dynamic algorithm also has constant update time (to keep track of kernels K_i), but has a query time of c^k (to compute X_i from K_i). Since in both cases our update times are constant and since it is not hard to see that one cannot improve the query times beyond the time needed by the fastest static algorithm, these bounds are optimal.

Main Result: A Fully Dynamic Hitting Set Kernel. In the fully dynamic case where edges may be inserted and deleted over time, the hypergraph may repeatedly switch between having and not having a size-k hitting set. This turns out to be a big obstacle for updating a kernel in just a few steps. Dynamic kernels have already been constructed by Alman, Mnich, and Williams [2]. They present a p_k-VERTEX-COVER kernel with $O(k)$ worst-case update time and $O(1)$ amortized update time. For the p_k-d-HITTING-SET they achieve a kernel of size $(d-1)!k(k+1)^{d-1}$ with update time $(d!)^d \cdot k^{O(d^2)}$.

In this paper, for each fixed number d we present a fully dynamic algorithm that maintains a p_k-d-HITTING-SET kernel of size $O(k^d)$ with constant update times.

▶ **Theorem 1.** *For every $d \geq 2$ there is a deterministic, fully dynamic kernel algorithm for the problem p_k-d-HITTING-SET that maintains at most $\sum_{i=0}^{d} k^i \leq (k+1)^d$ hyperedges in the kernel, has worst-case insertion time $3^d \operatorname{poly}(d)$, and worst-case deletion time $5^d \operatorname{poly}(d)$. In particular, as d is a constant, the dynamic kernel algorithm performs both insertions and deletions in time that is* constant *and* independent *of the input and parameter k.*

▶ **Corollary 2.** *There is a fully dynamic algorithm for p_k-d-HITTING-SET with update time $O(1)$ and query time $O(c^k)$, where $c = d - 1 + O(1/d)$.*

In order to achieve update times independent of k, this paper makes three major improvements on the general sunflower approach [1]. First, relevant objects are handled hierarchically. This allows an inductive construction and an analysis that improves the bounds on the kernel size as well as the update time. Second, we replace the notion of *strong edges* (see [2]) by *needed edges* to be defined later. Whenever a flower is formed, the replacement of its petals can be handled much more easily this way. Finally, the use of b-flowers (see also [17]) instead of generalized sunflowers [2] decreases the size of the kernel even further.

Our kernel is a *full kernel* in the sense of [11]: It does not just preserve a single size-k solution, but all of them. Therefore, we can use the kernel for counting and enumeration problems; and we can even use the whole kernel as approximate solution. The kernel size is optimal insofar as p_k-d-HITTING-SET has no kernel of size $O(k^{d-\epsilon})$ unless coNP $\subseteq$ NP/poly [13]. Notet that if we feed the hyperedges of a *static* hypergraph to our algorithm one-at-a-time, we compute a *static* hitting set kernel in time $3^d \operatorname{poly}(d) \cdot m$. Since the currently best algorithms for that tasks, see [4, 17, 30], run in time $2^d \operatorname{poly}(d) \cdot m$, our algorithm is not far from the best static runtime: the difference just lies in the constant factor 3^d versus 2^d.

Extension to Set Packing. Our kernelization can be adapted for p_k-MATCHING and the more general p_k-d-SET-PACKING: The input (H, k) is as before, but the question is whether there is a *packing* $P \subseteq E$ with $|P| \geq k$ (that is, $e \cap f = \emptyset$ for any two different $e, f \in P$).

▶ **Theorem 3.** *For every $d \geq 2$ there is a deterministic, fully dynamic kernel algorithm for the problem p_k-d-SET-PACKING that maintains at most $\sum_{i=0}^{d} (d(k-1))^i \leq d^d k^d$ hyperedges in the kernel, has worst-case insertion time $3^d \operatorname{poly}(d)$, and worst-case deletion time $5^d \operatorname{poly}(d)$.*

Related Work. Ever-better kernels for p_k-d-HITTING-SET are due to Flum and Grohe [20], van Bevern [30], and Fafianie and Kratsch [17]. Damaschke studied *full* kernels for the problem, which are kernels that contain all small solutions [11]. There are also optimized algorithms for specific values of d: for instance the algorithm by Buss and Goldsmith [7] for $d = 2$, or by Niedermeier and Rossmanith [28] and Abu-Khzam [1] for $d = 3$.

Dynamic algorithms can be used in a variety of monitoring applications such as maintaining a minimum spanning tree [22] or connected components [23]. There is also a recent trend in studying dynamic approximation algorithms, for instance for VERTEX-COVER [6]. Algorithms that maintain a solution for a dynamically changing input can also be studied using descriptive complexity, as suggested by Patnaik and Immerman [29]. A recent break-through result in this area is that reachability is contained in DynFO [12].

Iwata and Oka [24] were the first to combine kernelization and dynamic algorithms by studying a dynamic quadratic kernel for p_k-VERTEX-COVER. Their dynamic kernel algorithm requires $O(k^2)$ update time and works in a promise model where at all times it is guaranteed that there actually *is* a size-k vertex cover in the input graph. Alman, Mnich, and Williams extended this line of research by studying dynamic parameterized algorithms for a broad range of problems [2]. Among others, they provided a p_k-VERTEX-COVER kernel with $O(k)$ worst-case update time and $O(1)$ amortized update time that works in the fully dynamic model. Their generalization to a fully dynamic algorithm for p_k-d-HITTING-SET with a slightly larger kernel size and nononstant update time has already been mentioned above. Recent advances in dynamic FPT-algorithms where achieved by a dynamic data structure that maintains a optimum-height elimination forest for a graph of bounded treedepth [8].

Organization of This Paper. After a short introduction to dynamic algorithms, data structures, and parameterized complexity in Section 2, we first illustrate the algorithm for the special case of p_k-VERTEX-COVER in Section 3. Then, in Section 4, we generalize the algorithm to p_k-d-HITTING-SET. In Section 5 we argue that with slight modifications, the same algorithm can be used to maintain a polynomial kernel for p_k-d-SET-PACKING. In this publication we focus on explaining the core concepts – detailed proofs and technical details can be found in the technical report [3].

2 A Framework for Parametrized Dynamic Algorithms

Our aim is to *dynamically* maintain *kernels* with *minimal update time.* To formalize this, let us begin with the definition of *kernels* and then explain properties of *dynamic* kernels.

Parameterized Hypergraph Problems and Kernels. A *d-hypergraph* is a pair $H = (V, E)$ consisting of a set V of *vertices* and a set E of *hyperedges* with $e \subseteq V$ and $|e| \leq d$ for all $e \in E$. Let $n = |V|$ and $m = |E|$. The degree of v is $\deg_H(v) = \big|\{\, e \in E \mid v \in e \,\}\big|$. A *uniform d-hypergraph* has $|e| = d$ for all $e \in E$, e. g., a *graph* is a uniform 2-hypergraph. We use $\binom{V}{d}$ to denote the set $\{\, e \subseteq V \mid |e| = d \,\}$ of all size-d hyperedges and let $\binom{V}{\leq d} = \{\, e \subseteq V \mid |e| \leq d \,\}$. *Parameterized hypergraph problems* are sets $Q \subseteq \Sigma^* \times \mathbb{N}$, where *instances* $(H, k) \in \Sigma^* \times \mathbb{N}$ consist of a hypergraph H and a *parameter* k. A parameterized problem is in FPT if the question $(H, k) \in Q$ can be decided in time $f(k) \cdot (|V| + |E|)^c$ for some computable function f and constant c. It is known that $Q \in$ FPT holds iff *kernels* can be computed for Q in polynomial time [15]. Kernels of *polynomial size* are of special interest: For a polynomial σ, a *σ-kernel for an instance $(H, k) \in \Sigma^* \times \mathbb{N}$ of a problem Q* is another instance $(H', k') \in \Sigma^* \times \mathbb{N}$ with $|H'| \leq \sigma(k)$, $k' \leq \sigma(k)$, and $(H, k) \in Q \iff (H', k') \in Q$.

Dynamic Hypergraphs and Dynamic Kernels. One might consider several properties of a hypergraph that could change in a dynamic way. In this paper we consider as fixed and immutable the bound d on the hyperedge sizes, the vertex set V, and also the parameter k. That means only the most specific one, the hyperedge set E, will change dynamically. We assume that initially it is the empty set:

▶ **Definition 4** (Dynamic Hypergraphs). *A* dynamic hypergraph *consists of a fixed vertex set* $V = \{v_1, \dots, v_n\}$ *and a sequence* $o_1, o_2, o_3, \dots$ *of* update operations*, where each* o_j *is either* $\mathit{insert}(e_j)$ *or* $\mathit{delete}(e_j)$ *for a hyperedge* $e_j \subseteq V$*.*

A dynamic hypergraph defines a sequence of hypergraphs $H_0, H_1, \dots$ with $H_0 = (V, \emptyset)$, $H_j = (V, E(H_{j-1}) \cup \{e_j\})$ for $o_j = \mathit{insert}(e_j)$, and with $H_j = (V, E(H_{j-1}) \setminus \{e_j\})$ for $o_j = \mathit{delete}(e_j)$. For convenience (and without loss of generality) we assume only missing hyperedges are inserted and only existing ones deleted. A *dynamic* hypergraph algorithm gets the update sequence of a dynamic hypergraph as input and has to output a sequence of solutions, one for each H_i. Crucially, the solution for H_i must be generated before the next operation o_{i+1} is read. While after each update we could solve the problem from scratch for H_i, we may do better by taking into account that the difference between H_{i-1} and H_i is small. With the help of an internal *auxiliary data structure* A_i that the algorithm updates alongside the graphs, one might be able to solve the original problem faster after each update. *The problem we wish to solve dynamically* is to compute for each H_i a kernel K_i (as opposed to the problem of solving the parameterized problem Q itself).

▶ **Definition 5** (Dynamic Kernel Algorithm). *Let* Q *be a parameterized problem and* σ *a polynomial. A* dynamic kernel algorithm Algo for Q with kernel size $\sigma(k)$ *has three methods:*

1. Algo.$\mathit{init}(n, k)$ *gets the size* n *of* V *and the parameter* k *as inputs, neither of which will change during a run of the algorithm, and must initialize an auxiliary data structure* A_0 *and a kernel* K_0 *for* (H_0, k) *and* Q *and* σ *(observe that* $H_0 = (V, \emptyset)$ *holds).*
2. Algo.$\mathit{insert}(e)$ *gets a hyperedge* e *to be added to* H_{i-1} *and must update* A_{i-1} *and* K_{i-1} *to* A_i *and* K_i *with, again,* K_i *being a kernel for* (H_i, k) *and* Q *and* σ*.*
3. Algo.$\mathit{delete}(e)$ *removes an edge instead of adding it.*

One could also require that only the data structure A_i is updated in each step, while a kernel K_i would only be needed to be computed upon a query request. This would allow to differentiate between update times and query times for computing kernels. By requiring that the kernel K_i is explicitly computed at each step alongside A_i, our definition implies a query time of zero for computing K_i. However, solving the query $(H_i, k) \in Q$ using K_i may take exponential time in k. Concerning the update times, an efficient dynamic kernel algorithm should of course compute A_i and K_i faster than a static kernelization that processes H_i completely. The best one could hope for is constant time for the initialization and per update, even independent of the parameter k – and this is exactly what we achieve in this paper.

Data Structures for Dynamic Algorithms. The A_i rely on data structures such as objects and arrays. We additionally use a novel data structure called *relevance list,* which are ordinary lists equipped with a *relevance bound* $\rho \in \mathbb{N}$*:* the first ρ elements are said to be *relevant,* while the others are *irrelevant.* This data structure supports insertion and deletion, querying the relevance status of an element, and querying the last relevant element – each in $O(1)$ time. For concrete implementations and an analysis, please see the technical report [3].

3 Dynamic Vertex Cover with Constant Update Time

In order to better explain the ideas behind our dynamic kernel algorithm, we first tackle the case $d = 2$ in this section and show how we can maintain kernels of size $O(k^2)$ for the vertex cover problem with update time $O(1)$. The idea is based on a well-known *static* kernel: Buss [7] noticed that in order to cover all edges of a graph $G = (V, E)$ with k vertices, we *must* pick any vertex with more than k neighbors (let us call such vertices *heavy*). If there are more than k^2 edges after all heavy vertices have been picked and removed, no vertex cover of size k is possible (since each light vertex can cover at most k edges).

To turn this idea into a dynamic kernel, let us first consider only insertions. Initially, new edges can simply be added to the kernel; but at some point a vertex v "becomes heavy." In the static setting one would remove v from the graph and decrease the parameter by 1. In the dynamic setting, however, removing v with its adjacent edges would take time $O(k)$ rather than $O(1)$. Instead, we leave v in the graph, but do *not* add further edges containing v to the kernel once v becomes heavy. We call the first $k + 1$ edges *relevant for the vertex* and the rest *irrelevant*. By putting the relevant edges of a heavy vertex in the kernel, we ensure that this vertex still must be chosen for any vertex cover. By leaving out the irrelevant edges, we ensure a kernel size of at most $O(k^2)$. More precisely, if the kernel size now threatens to exceed $k^2 + k + 1$, then any additional edges will be *irrelevant for the kernel* since the already inserted edges already form a proof that no size-k vertex cover exists.

Being relevant for a vertex is a "local" property: For an edge $e = \{u, v\}$, the vertex u may consider e to be relevant, while v may consider it to be irrelevant. An edge only "makes it to the kernel" when it is relevant for both endpoints – then it will be called *needed*. It is not obvious that this is how the case of a "disagreement" should be resolved and that this is the right notion of "needed edges" – but Lemma 8 shows that it leads to a correct kernel.

A Dynamic Vertex Cover Kernel Algorithm. We now turn the sketched ideas into a formal algorithm in the sense of Definition 5. The initialization sets up the auxiliary data structures: One relevance list L_v per vertex v to keep track of the edges that are relevant for v and one relevance list L to keep track of the edges that are relevant for the kernel. The code violates the requirement that the *initialization procedures* should run in constant time, but a known code transformation [27] for ensuring this will be discussed in the general hitting set case.

```
1  method DynKernelVC.init(n, k) // V = {v_1, ..., v_n} holds by definition
2    for v ∈ V do
3      L_v ← new relevance list(k + 1) // Keep track of relevant edges for a vertex
4    L ← new relevance list(k^2 + k + 1)  // Keep track of relevant edges for the kernel
```

The insert operation adds an edge e to the relevance lists of both endpoints of e. Furthermore, it also adds e to L if it is *needed,* which meant "relevant for both sides".

```
5  method DynKernelVC.insert(e)
6    L_u.append(e); L_v.append(e)
7    check if needed(e)
8
9  function check if needed(e) // assume e = {u, v}
10   if L_u.is relevant(e) ∧ L_v.is relevant(e) then
11     L.append(e)
```

The delete operation for an edge e is more complex: When $e = \{u, v\}$ is removed from the lists L_u, L_v, and L, formerly irrelevant edges may suddenly become relevant from the point of view of these three lists and, thus, possibly also needed. Fortunately, we know which

edge e' may suddenly have become relevant for a list: After the removal of e, the edge e' that is now the last relevant edge stored in the list is the (only) one that may have become relevant – and relevance lists keep track of the last relevant element.

```
12  method DynKernelVC.delete(e) // assume e = {u, v}
13    L.delete(e)
14    L_u.delete(e); L_v.delete(e)
15    check if needed(L_u.last relevant); check if needed(L_v.last relevant)
```

Correctness and Kernel Size. The relevant edges in L clearly have some properties that we would expect of a kernel: First, there are at most $k^2 + k + 1$ of them (for the simple reason that L caps the number of relevant edges in line 4) – which is exactly the size that a kernel should have. Second, it is also easy to see from the code of the algorithm that all operations run in time $O(1)$. Two lemmas make these observations precise, where $R(L)$ denotes the set of relevant edges in a list L and $E(L)$ denotes all edges in L; and where we say that a dynamic algorithm *maintains an invariant* if that invariant holds for its auxiliary data structure right after the *init* method has been called and after every call to *insert* and *delete*.

▶ **Lemma 6.** DynKernelVC *maintains the invariant* $|R(L)| \leq k^2 + k + 1$.

▶ **Lemma 7.** DynKernelVC.*insert and* DynKernelVC.*delete run in time* $O(1)$.

The crucial, much less obvious property of the algorithm is stated in the next lemma, whose proof contains a non-trivial recursive analysis showing that irrelevant edges must already be covered by relevant edges inserted earlier.

▶ **Lemma 8.** DynKernelVC *maintains the invariant that* $(V, R(L))$ *and the current graph* (V, E) *have the same size-*k *vertex covers.*

Put together, we get the following special case of Theorem 1:

▶ **Theorem 9.** DynKernelVC *is a dynamic kernel algorithm for* p_k-VERTEX-COVER *with update time* $O(1)$ *and kernel size* $k^2 + k + 1$.

Proof. Lemmas 6, 7, and 8 together state that at all times during a run of the algorithm DynKernelVC the graph $(V, R(L))$ has at most $k^2 + k + 1$ edges and has the same size-k vertex covers as the current graph. Thus, $(V, R(L))$ is *almost* a kernel *except* that $R(L)$ is actually a linked list of edges (with potentially large vertex identifiers).

However, we can simultaneously keep track of an adjacency matrix of a graph K with the vertex set $V_K = \{1, \ldots, 2(k^2 + k + 1)\}$ and with an edge set E_K that is always isomorphic to $R(L)$, that is, $E_K \sim R(L)$. In particular, K has a size-k hitting set if, and only if, G has one. See [3] for technical details.

The update times are constant. The time needed for DynKernelVC.*init*(n, k) can be made constant with a special new-initialized-with construct based on a technique in [27] as already mentioned and discussed in more detail below. ◀

4 Dynamic Hitting Set Kernels

The hitting set problem is a generalization of VERTEX-COVER to hypergraphs. However, allowing larger hyperedges introduces considerable complications into the algorithmic machinery. Nevertheless, we still seek and prove an update time that is constant. More precisely, it is independent of $n = |V|$, $m = |E|$, and the parameter k, while it does depend on d (in

fact even exponentially). Such an exponential dependency on d seems currently unavoidable, since a direct consequence of our dynamic algorithm is a static algorithm with running time $3^d \operatorname{poly}(d) \cdot m$, and the currently best static algorithm runs in time $2^d \operatorname{poly}(d) \cdot m$.

The first core idea of our algorithm concerns a replacement notion for the "heavy vertices" from the previous section. *Sunflowers* [16] are usually a stand-in (see [20, Section 9.1] and [32, 5]), but they are hard to find and especially hard to manage dynamically. Instead, we use an idea first proposed by Fafianie and Kratsch [17], but adapted to our dynamic setting: a generalizations of sunflowers, which we call *b-flowers* for different parameters $b \in \mathbb{N}$ that will be easier to keep track of dynamically.

The second core idea is to recursively reduce each case d to the case $d-1$: For a fixed $d > 2$, we compute a set of hyperedges relevant for the kernel (the set $R(L)$, but now called $R(L^d[\emptyset])$ in the more general case), but *additionally* we dynamically keep track of an instance for p_k-$(d-1)$-HITTING-SET and merge the dynamic kernel for this instance (which we get from the recursion) with the list of hyperedges relevant for the kernel.

4.1 From High-Degree Vertices in Graphs to Flowers in Hypergraphs

A *sunflower* in a d-hypergraph $H = (V, E)$ is a collection of hyperedges $S \subseteq E$ such that there is a set $c \subseteq V$, called the *core*, with $x \cap y = c$ for all distinct pairs $x, y \in S$. For example, the edges adjacent to a heavy vertex v form a (large) sunflower with core $\{v\}$. In general, any size-k hitting set has to intersect with the core of a sunflower if it has more than k edges – which means that replacing large sunflowers by their cores is a reduction rule for p_k-d-HITTING-SET. This rule yields a kernel since the *Sunflower Lemma* [16] states that every d-hypergraph with more than $k^d \cdot d!$ hyperedges contains a sunflower of size $k+1$.

Unfortunately, it is not easy to find sunflowers for larger d in the first place, let alone to keep track of them in a dynamic setting with constant update times. Rather than trying to find all sunflowers, we use a more general concept called *b-flowers.*

▶ **Definition 10.** *For a hypergraph $H = (V, E)$ and $b \in \mathbb{N}$, a b-*flower with core *c is a set $F \subseteq E$ such that $c \subseteq e$ for all $e \in F$ and $\deg_{(V,F)}(v) \leq b$ for all $v \in V - c$.*

Note that a 1-flower is exactly a sunflower and, thus, b-flowers are in fact a generalization of sunflowers, see Figure 1 for an example.

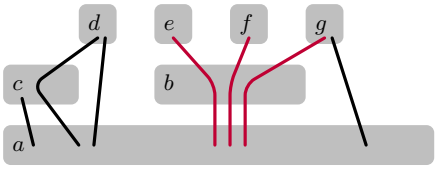


Figure 1 A hypergraph $H = (\{a, b, c, d, e, f, g\}, E)$ in which each hyperedge $e \in E$ is drawn as a line and contains all vertices it "touches". The three red edges form a 1-flower (a sunflower) with core $\{a, b\}$. The hyperedges $\big\{\{a, c\}, \{a, d\}, \{a, g\}, \{a, b, e\}\big\}$ also form a 1-flower, now with with core $\{a\}$, but if we add the hyperedges $\{a, b, f\}$ and $\{a, c, d\}$, we no longer have a 1-flower – but still a 2-flower with core $\{a\}$. All edges together form a 3-flower with core $\{a\}$.

▶ **Lemma 11.** *Let F be a b-flower with core c in H and X a size-k hitting set of H. If $|F| > b \cdot k$, then $X \cap c \neq \emptyset$ ("X must hit c").*

Proof. If we had $X \cap c = \emptyset$, then each $v \in X$ could hit at most b hyperedges in F since $\deg_{(V,F)}(v) \leq b$. Then F can contain at most $b \cdot |X|$ hyperedges, contradicting $|F| > b \cdot k$. ◀

4.2 Dynamic Hitting Set Kernels: A Recursive Approach

As previously mentioned, the core idea behind our main algorithm is to recursively reduce the case d to $d-1$. To better explain this idea, we illustrate how the (already covered) case $d=2$ can be reduced to $d=1$ and how this in turn can be reduced to $d=0$. Following this, we present the complete recursive algorithm, prove its correctness, and analyze its runtime.

Recall that DynKernelVC adds up to $k+1$ edges per vertex v into the kernel $R(L)$ to ensure that v "gets hit." In the recursive hitting set scenario we ensure this differently: When we notice that v is "forced" into all hitting sets, we add a new hyperedge $\{v\}$ to an internal 1-hypergraph used exclusively to keep track of the forced vertices (clearly the only way to hit $\{v\}$ is to include v in the hitting set). When, later on after a deletion, we notice that a singleton hyperedge is no longer forced, we remove it from the internal 1-hypergraph once more. Since we have to ensure that not too many new hyperedges make it into the final kernel, we keep track of a *dynamic kernel of the internal 1-hypergraph* (using a dynamic hitting set algorithm for $d=1$) and then *join* this kernel with $R(L)$.

Using a hypergraph to track the forced vertices allows us to change the relevance bounds of the algorithm: For the lists L_v these were $k+1$, but since we explicitly "force" $\{v\}$ into the solution by generating a new hyperedge, it is enough to set the bound to k. Similarly, the bound for the original list L was set to k^2+k+1 since this constitutes a proof that no size-k vertex cover exists. In the new setting with the relevance bound for L_v lowered to k, we can also lower the relevance bound for L to k^2: All vertices $v \in V$ have a degree of at most k in $R(L)$ and, thus, k vertices can hit at most k^2 hyperedges. If L contains more elements, we consider the (unhittable) empty hyperedge as *forced* and add it to the 1-hypergraph.

In order to dynamically keep track of a kernel for the internal 1-hypergraph, we proceed similarly: We simply put all its hyperedges (which have size 1 or 0) in a list (called $L^1[\emptyset]$ in the algorithm). If the number of hyperedges in this list exceeds k, we immediately know that no hitting set of size k exists; and we "recursively remember this" by inserting the empty set into yet another internal 0-hypergraph – this is the recursive call to $d=0$.

Managing Needed and Forced Hyperedges. In the general setting (now for arbitrary d), we need a uniform way to keep track of lists like the L_v and L for the many different internal hypergraphs. We do this using arrays L^i for $i \in \{0,\dots,d\}$ with domains $\binom{V}{\leq i}$, one for each i-hypergraph, where each $L^i[s]$ stores a relevance list. The list $L^i[s]$ has relevance bound $k^{i-|s|}$ and we only stores edges $e \in \binom{V}{\leq i}$ with $e \supseteq s$ in it.

The idea behind this construction is as follows. For $d=2$ the list $L^2[\{v\}]$ represents the list L_v of DynKernelVC and $L^2[\emptyset]$ represents the list L. The lists $L^2[\{u,v\}]$ are new and will only store a single element and are only added to simplify the code: When an edge $e=\{u,v\}$ is inserted into the 2-hypergraph, we add it to $L^2[e]$, but more importantly also to $L^2[\{u\}]$ and $L^2[\{v\}]$. If it is *relevant* for both lists, we call it *needed* and add it also to $L^2[\emptyset]$. If $L^2[s]$ contains an irrelevant edge, then s is *forced,* and we insert it into $L^1[s]$. For L^1, the array that manages the internal 1-hypergraph, we have similar rules for being needed and forced. An example of how this works is shown in Figure 2. The next two definitions generalize the idea of *needed* and *forced* hyperedges to arbitrary d and lie at the heart of our algorithm. The earlier rules for $d=2$ are easily seen to be special cases:

▶ **Definition 12** (Needed Hyperedges and the Need Invariant). *A hyperedge e is* needed *in a list $L^i[s]$ with $s \subsetneq e$ if $e \in R(L^i[t])$ holds for all $t \subseteq e$ with $s \subsetneq t$. A dynamic algorithm maintains the* Need Invariant *if for all $e \in \binom{V}{\leq d}$, all $s \subsetneq e$, and all $i \in \{0,\dots,d\}$, the list $L^i[s]$ contains e iff e is needed in it.*

▶ **Definition 13** (Forced Hyperedges and the Force Invariant). *A set of vertices s is* forced by $L^i[s]$ into $L^{i-1}[s]$ *or just* forced by $L^i[s]$ *if $L^i[s]$ has an irrelevant hyperedge. A dynamic algorithm maintains the* Force Invariant *if for all $i \in \{1, \ldots, d\}$ and all $s \in \binom{V}{<i}$, the list $L^{i-1}[s]$ contains s iff s is forced by $L^i[s]$.*

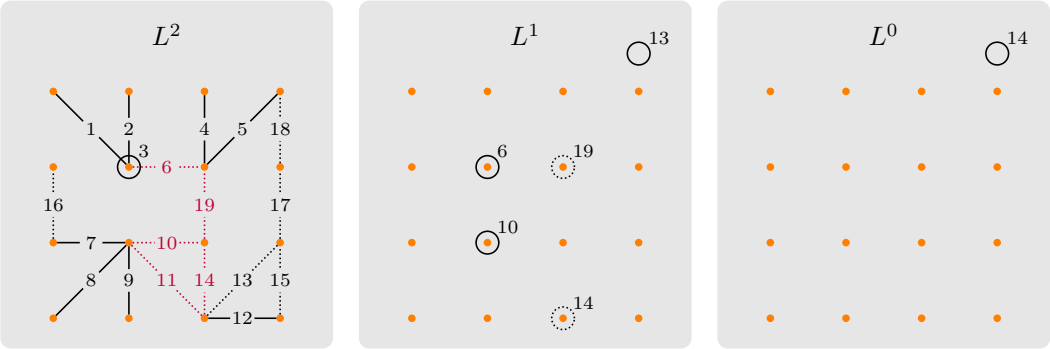


Figure 2 The data stored in the lists L^2, L^1, and L^0 for $k = 3$ and a dynamic 2-hypergraph with 16 (orange) vertices created with 19 edge insertions (numbers indicate insertion times; there are no deletions in this example). Normal edges are shown as straight lines, singleton edges $\{v\}$ as circles around v, and the empty set as an empty circle. In L^2, the members of $L^2[\emptyset]$ are drawn in black. They are all relevant for both endpoints and thus needed in $L^2[\emptyset]$. The red edges are not relevant for one of the endpoints and thus neither needed in nor added to $L^2[\emptyset]$. Among the black edges, only the first $k^2 = 9$ are relevant, the rest (dotted) are irrelevant. In L^1, we store the "forced s" that L^2 forces into L^1 at the indicated timestamps: each time, it is the first time an irrelevant edge e is inserted into $L^2[s]$. After the first three s (two singletons at timestamps 6 and 10 and then the empty set at timestamp 13) got inserted into $L^1[\emptyset]$, further edges are irrelevant and trigger the insertion of the empty set into $L^0[\emptyset]$.

We will show in Lemmas 18 and 21 that the union $K = \bigcup_{i=0}^{d} R(L^i[\emptyset])$ is the sought kernel: Each $R(L^i[\emptyset])$ contains (only) those hyperedges e that have not already been taken care of by having forced a subset s of e into the internal $(i-1)$-hypergraph.

In the following, we develop code that ensures that the Need Invariant and the Force Invariant hold at all times. We will show that this is the case both for an insert operation and also for delete operations. Then we show that the invariants imply that $K = \bigcup_{i=0}^{d} R(L^i[\emptyset])$ is a kernel for the hitting set problem. Finally, we analyze the runtimes.

Initializiation. The initialization creates the arrays L^i and the relevance lists.

```
1  method DynKernelHS.init(n, k, d)
2    // Keep track of relevant edges per vertex (V = {v_1, ..., v_n} holds by definition):
3    for i ∈ {0, ..., d} do
4      L^i ← new ARRAY(binom(V, ≤i)) initialized with
5                                  (new RELEVANCE LIST(k^{i−|s|})) for s ∈ binom(V, ≤i)
```

The construct *new* ARRAY(D) *initialized with* $f(s)$ *for* $s \in D$ allocates a new array with domain D and then immediately set the value of each entry $s \in D$ to $f(s)$. (So, in our case, each $L^i[s]$ will be a new, empty relevance list with relevance bound $k^{i-|s|}$.) The important point is that both allocation and pre-filling can be done in *constant* time using the standard trick to work with uninitialized memory [27].

Independently of the *time* needed for the allocation, observe that the amount of memory we allocate is about $O(n^d)$ – which is already too much in almost any practical setting for $d = 3$, see [31, Chapter 5] for a discussion of experimental findings. However, we will only

use a very small fraction of the allocated memory: The only lists $L^i[s]$ that are non-empty at any point during a run of the algorithms are those where $s \subseteq e \in E$ holds. This means that we actually only need space $O(2^d|E|)$ to manage the non-empty lists if we use hash tables. Of course, this entails a typically non-constant overhead per access for managing the hash tables, which is why our analysis is only for the wasteful implementation above. For a clever way around this problem in the *static* setting, see [32].

▶ **Lemma 14.** *The Need and Force Invariant hold after the init method has been called.*

Proof. All lists are empty after the initialization. ◀

Insertions. We view insertions as a special case of "forcing an edge," namely as forcing it into the lists of L^d. Adding an edge e to a list $L^i[e]$ can, of course, change the set of relevant edges in $L^i[e]$, which means that e may also be needed in lists $L^i[s]$ for $s \subsetneq e$. It is the job of the method *fix needs downward* to add e to the necessary lists.

```
6  method DynKernelHS.insert(e)
7    call insert(e, d) // The hyperedges of H always get inserted into L^d
8
9  function insert(s, i)
10   if L^i[s] does not already contain s then // Sanity check
11     L^i[s].append(s) // s is always needed in L^i[s]
12     call fix force(s, i)
13     call fix needs downward(s, s, i)
14
15 function fix needs downward(s, p, i)
16   // Ensure that the Need Invariant holds for s with respect to all L^i[s'] with s' ⊆ p,
17   // assuming that the Need Invariant holds for s with respect to all L^i[s*] with s* ⊇ p:
18   for s' ⊊ p in decreasing order of size do // Add s to all L^i[s'] where s is needed
19     if L^i[s'] does not contain s then // Sanity check
20       if ∀v ∈ p − s' : s ∈ R(L^i[s' ∪ {v}]) then // Is s needed for L^i[s']?
21         L^i[s'].append(s) // Yes: it is relevant for all its direct and hence all its supersets
22         call fix force(s', i)
23
24 function fix force(s, i)
25   if L^i[s].has irrelevant elements then // Is s forced?
26     call insert(s, i − 1)
```

The method *fix needs downward* is more complex than necessary here, but we will need the extra flexibility for the delete method later on: For two sets of vertices s and p with $s \supseteq p$ and a fixed number i, let us say that *the Need Invariant holds for s above p* if for all $s' \supseteq p$ we have $s \in E(L^i[s'])$ iff s is needed for $L^i[s']$. Let us say that *the Need Invariant holds for s below s'* if for all $s' \subseteq p$ we have $s \in E(L^i[s'])$ iff s is needed for $L^i[s']$. In the context of the insert operation, *fix needs downward* always gets called with $s = p$, meaning that in the following lemma the premise ("the Need Invariant holds for s above p") is trivially true.

▶ **Lemma 15.** *Let s and p with $s \supseteq p$ be sets of vertices and let i be fixed. Suppose the Need Invariant holds for s above p. Then after the call fix needs downward(s, p, i) the Need Invariant will also hold for s' below p.*

Proof. We need to show that the code ensures for all $s' \subseteq p$ that if s is needed in $L^i[s']$, it gets inserted. It is the job of line 20 to test whether such an insertion is necessary. The line tests whether $\forall v \in p - s' : s \in R(L^i[s' \cup \{v\}])$ holds. By Definition 12 of needed hyperedges,

Table 1 Handling of an insertion for $d = 3$ and $k = 2$. The upper part shows for selected relevance lists a snapshot of their relevant elements (left), their irrelevant elements (right), the list lengths, and the relevance bounds. In lines $L^i[s]$ where the length exceeds the bound (in red), s is forced into $L^{i-1}[s]$. The insertion of $e = \{u, v, w\}$ triggers: (i) e is added to the list $L^3[e]$; (ii) since e is relevant in $L^3[e]$, it is added to the lists for $\{u, v\}$, $\{u, w\}$, and $\{v, w\}$ as well; (iii) e becomes needed in $L^3[\{u\}]$ and gets inserted; (iv) since $L^3[\{u\}]$ was already at maximum capacity ($k^2 = 4$), e becomes the first irrelevant element in this list; (v) this forces $\{u\}$ into $L^2[\{u\}]$; (vi) there $\{u\}$ is the first element and hence relevant and also needed in $L^2[\emptyset]$, where it gets inserted.

$E(L^i[s]) =$	$R(L^i[s])$	$\cup$	$E(L^i[s]) \setminus R(L^i[s])$	size $\leq$ bound?
$E(L^3[\{u,v,w\}]) =$	$\emptyset$	$\cup$	$\emptyset$	$0 \leq k^0 = 1$
$E(L^3[\{u,v\}]) =$	$\{\{u,v,x\}\}$	$\cup$	$\emptyset$	$1 \leq k^1 = 2$
$E(L^3[\{v\}]) =$	$\{\{u,v,x\}\}$	$\cup$	$\emptyset$	$1 \leq k^2 = 4$
$E(L^3[\{u\}]) =$	$\{\{u,v,x\},\{u\},\{u,x,y\},\{u,z\}\}$	$\cup$	$\emptyset$	$4 \leq k^2 = 4$
$E(L^3[\emptyset]) =$	$\{\{u,v,x\},\{u\},\{u,x,y\},\{u,z\}\}$	$\cup$	$\emptyset$	$4 \leq k^3 = 8$
$E(L^2[\{u\}]) =$	$\emptyset$	$\cup$	$\emptyset$	$0 \leq k^1 = 2$
$E(L^2[\emptyset]) =$	$\emptyset$	$\cup$	$\emptyset$	$0 \leq k^2 = 4$
Insertion of $e = \{u, v, w\}$ now yields:				
$E(L^3[\{u,v,w\}]) =$	$\{\{u,v,w\}\}$	$\cup$	$\emptyset$	$1 \leq k^0 = 1$
$E(L^3[\{u,v\}]) =$	$\{\{u,v,x\}\{u,v,w\}\}$	$\cup$	$\emptyset$	$2 \leq k^1 = 2$
$E(L^3[\{v\}]) =$	$\{\{u,v,x\}\{u,v,w\}\}$	$\cup$	$\emptyset$	$2 \leq k^2 = 4$
$E(L^3[\{u\}]) =$	$\{\{u,v,x\},\{u\},\{u,x,y\},\{u,z\}\}$	$\cup$	$\{\{u,v,w\}\}$	$5 > k^2 = 4$
$E(L^3[\emptyset]) =$	$\{\{u,v,x\},\{u\},\{u,x,y\},\{u,z\}\}$	$\cup$	$\emptyset$	$4 \leq k^3 = 8$
$E(L^2[\{u\}]) =$	$\{\{u\}\}$	$\cup$	$\emptyset$	$1 \leq k^1 = 2$
$E(L^2[\emptyset])=$	$\{\{u\}\}$	$\cup$	$\emptyset$	$1 \leq k^2 = 4$

what we are *supposed* to test is whether for all $t \subseteq s$ with $s' \subsetneq t$ we have $s \in R(L^i[t])$. Observe that the property of being needed is "upward closed": if s is needed in $L^i[p]$, it is also needed in all $L^i[s^*]$ with $p \subseteq s^* \subseteq s$. This implies that by processing the hyperedges s' in descending order of size (line 18), s will be needed for $L^i[s']$ iff s is needed for all the hyperedges $t = s' \cup \{v\}$ that are one element larger than s. This is exactly what we test. ◀

▶ **Lemma 16.** *The Need and Force Invariant are maintained by the insert method.*

Proof. For the Need Invariant, observe that whenever the *fix force* method adds an edge s to $L^i[s]$ in line 11, it also calls *fix needs*(s, s, i) right away. By Lemma 15, this ensures that s is inserted exactly into those $L^i[s']$ for $s' \subseteq s$ where it is needed. For the Force Invariant, observe that we only *add* elements to lists of L^i, which means that they can only *become* forced – they cannot lose this status through an addition of an edge. However, after any insertion of s into any list of L^i (namely, in lines 11 and 21) we immediately call *fix forced*, which inserts s into $L^{i-1}[s]$ if s is forced. ◀

Deletions. The delete operation has to delete an edge e from all places where it might have been inserted to, which is just from all lists $L^d[s]$ for $s \subseteq e$. However, removing e from such a list can have two side-effects: First, it can cause $L^d[s]$ to lose its last irrelevant element, changing the status of s from "forced" to "not forced" and we need to "unforce" it (remove it from $L^{d-1}[s]$), which may recursively entail new deletions. Furthermore, removing e

from $L^d[s]$ may make a previous irrelevant hyperedge (the first irrelevant hyperedge of $L^d[s]$) relevant. Then one has to fix the needs for this hyperedge once more, which may entail new inserts and forcings, but no new deletions (see Table 2 for an example).

```
27 method DynKernelHS.delete(e)
28   call delete(e, d)
29
30 function delete(s, i)
31   if L^i[s] contains s then // Sanity check
32     // Delete s and subsets of s if no longer forced
33     for s' ⊆ s do
34       L^i[s'].delete(s) // Delete e from all lists that could contain it
35       if not L^i[s'].has irrelevant elements then // Has s' now lost its forced status?
36         call delete(s', i − 1)
37
38     // Restore Need Invariant for hyperedges that have suddenly become relevant
39     for s' ⊆ s do
40       f ← L^i[s'].last relevant
41       call fix needs downward(f, s', i) // (Only) the last relevant may have changed
```

Table 2 For the situation illustrated in the upper part, we delete the edge $e = \{u, v, w\}$. This triggers: (i) e gets deleted from all $L^3[s]$ with $s \subseteq e$; (ii) $\{u, v, z\}$ becomes relevant for $\{u, v\}$ in L^3; (iii) since that was the last irrelevant edge for the set $\{u, v\}$, the edge $\{u, v\}$ gets deleted from the graph represented by L^2; (iv) $\{u, z\}$ becomes relevant for $\{u\}$ in L^2; (v) as this was the last irrelevant edge, $\{u\}$ gets deleted from L^1; (vi) $\{u, z\}$ becomes relevant for $\{u\}$ and needed for $L^2[\emptyset]$; (vii) $\{u, v, z\}$ is now also needed in $L^3[\{u\}]$ and, thus, in $L^3[\emptyset]$ as well.

$E(L^i[s]) =$	$R(L^i[s])$	$\cup$	$E(L^i[s]) \setminus R(L^i[s])$	size $\leq$ bound?
$E(L^3[\{u,v\}]) =$	$\{\{u,v,y\},\{u,v,w\}\}$	$\cup$	$\{\{u,v,z\}\}$	$3 > k^1 = 2$
$E(L^3[\{u,y\}]) =$	$\{\{u,y,v\},\{u,y,z\}\}$	$\cup$	$\{\{u,y,x\}\}$	$3 > k^1 = 2$
$E(L^3[\{u,z\}]) =$	$\{\{u,z,v\},\{u,z,r\}\}$	$\cup$	$\{\{u,z,y\}\}$	$3 > k^1 = 2$
$E(L^3[\{u\}]) =$	$\{\{u,y,v\},\{u,v,w\},\{u,z,r\}\}$	$\cup$	$\emptyset$	$3 \leq k^2 = 4$
$E(L^3[\emptyset]) =$	$\{\{u,y,v\},\{u,v,w\},\{u,z,r\}\}$	$\cup$	$\emptyset$	$3 \leq k^3 = 8$
$E(L^2[\{u\}]) =$	$\{\{u,v\},\{u,y\}\}$	$\cup$	$\{\{u,z\}\}$	$3 > k^1 = 2$
$E(L^2[\emptyset]) =$	$\{\{u,v\},\{u,y\}\}$	$\cup$	$\emptyset$	$2 \leq k^2 = 4$
$E(L^1[\{u\}]) =$	$\{\{u\}\}$	$\cup$	$\emptyset$	$1 \leq k^0 = 1$
$E(L^1[\emptyset]) =$	$\{\{u\}\}$	$\cup$	$\emptyset$	$1 \leq k^1 = 2$
Deletion of $e = \{u, v, w\}$ now yields:				
$E(L^3[\{u,v\}]) =$	$\{\{u,v,y\},\{u,v,z\}\}$	$\cup$	$\emptyset$	$2 \leq k^1 = 2$
$E(L^3[\{u,y\}]) =$	$\{\{u,y,v\},\{u,y,z\}\}$	$\cup$	$\{\{u,y,x\}\}$	$3 > k^1 = 2$
$E(L^3[\{u,z\}]) =$	$\{\{u,z,v\},\{u,z,r\}\}$	$\cup$	$\{\{u,z,y\}\}$	$3 > k^1 = 2$
$E(L^3[\{u\}]) =$	$\{\{u,y,v\},\{u,z,r\},\{u,v,z\}\}$	$\cup$	$\emptyset$	$3 \leq k^2 = 4$
$E(L^3[\emptyset]) =$	$\{\{u,y,v\},\{u,z,r\},\{u,v,z\}\}$	$\cup$	$\emptyset$	$3 \leq k^3 = 8$
$E(L^2[\{u\}]) =$	$\{\{u,y\},\{u,z\}\}$	$\cup$	$\emptyset$	$2 \leq k^1 = 2$
$E(L^2[\emptyset]) =$	$\{\{u,y\},\{u,z\}\}$	$\cup$	$\emptyset$	$2 \leq k^2 = 4$
$E(L^1[\{u\}]) =$	$\emptyset$	$\cup$	$\emptyset$	$0 \leq k^0 = 1$
$E(L^1[\emptyset]) =$	$\emptyset$	$\cup$	$\emptyset$	$0 \leq k^1 = 2$

▶ **Lemma 17.** *The Need and Force Invariant are maintained by the delete method.*

Kernel. As stated earlier, the dynamic kernel maintained by DynKernelHS is the set $K = \bigcup_{i=0}^{d} R(L^i[\emptyset])$. ($K$ is given only indirectly via d linked lists, but one can do the same transformations as in the proof of Theorem 9 to obtain a compact matrix representation.)

Correctness. We have already established that the algorithm maintains the Need Invariant and the Force Invariant. Our objective is now to show that DynKernelHS does, indeed, maintain a kernel at all times. We start with the size:

▶ **Lemma 18.** DynKernelHS *maintains the invariant* $|K| \leq k^d + k^{d-1} + \cdots + k + 1$.

Proof. The *init*-method installs a relevance bound of k^i for $L^i[\emptyset]$ for all $i \in \{0, \ldots, d\}$. ◀

Lemma 21 shows the crucial property that the current K has a hitting set of size k iff the current hypergraph does. The proof hinges on the following two lemmas on "flower properties":

▶ **Lemma 19.** DynKernelHS *maintains the invariant that for all* $i \in \{0, \ldots, d\}$ *and all* $s \in \binom{V}{\leq i-1}$, *the set* $E(L^i[s])$ *is a* $k^{i-|s|-1}$*-flower with core* s.

Proof. First, for all $e \in E(L^i[s])$ we have $s \subseteq e$ since in all places in the *insert*-method where we append an edge e to a list $L^i[s]$, we have $s \subseteq e$ (in line 11 we have $e = s$ and in line 21 we have $s \subsetneq e$ by line 18). Second, consider a vertex $v \in V - s$. We have to show that $\deg_{(V,E(L^i[s]))}(v) \leq k^{i-|s|-1}$ (recall Definition 10) or, spelled out, that v lies in at most $k^{i-|s|-1}$ hyperedges $e \in E(L^i[s])$. By the Need Invariant, all $e \in E(L^i[s])$ are needed. In particular, for $t = s \cup \{v\}$ Definition 12 tells us $e \in R(L^i[t])$. Therefore, we have $\{\, e \in E(L^i[s]) \mid v \in e \,\} \subseteq R(L^i[s \cup \{v\}])$ and the latter set has a maximum size of $k^{i-|s\cup\{v\}|} = k^{i-|s|-1}$ due to the relevance bound installed in line 5. ◀

▶ **Lemma 20.** DynKernelHS *maintains the invariant that for all* $X \in \binom{V}{\leq k}$ *and for all* $i \in \{1, \ldots, d\}$ *and all* $s \in \binom{V}{\leq i}$, *if* s *is forced into* L^{i-1} *and if* X *hits all elements of* $E(L^i[s])$, *then* X *hits* s.

Proof. By Definition 13, "being forced into L^{i-1}" means that $L^i[s]$ has an irrelevant edge. In particular, $|E(L^i[s])| > k^{i-|s|}$. By Lemma 19, $E(L^i[s])$ is a $k^{i-|s|-1}$-flower with core s. By Lemma 11, since $|E(L^i[s])| > k^{i-|s|} = k \cdot k^{i-|s|-1}$, we know that X hits s, as claimed. ◀

▶ **Lemma 21.** DynKernelHS *maintains the invariant that* H *and* K *have the same size-*k *hitting sets.*

Run-Time Analysis. It remains to bound the run-times of the insert and delete operations.

▶ **Lemma 22.** DynKernelHS.*insert*(e) *runs in time* $3^d \operatorname{poly}(d)$.

Proof. The call DynKernelHS.*insert*(e) will result in at least one call of *insert*(s, i): The initial call is for $s = e$ and $i = d$, but the method *fix force* may cause further calls for different values. However, observe that *all* subsequently triggered calls have the property $s \subsetneq e$ and $i < d$. Furthermore, observe that *insert*(s, i) returns immediately if s has already been inserted. We will establish a time bound $t_{\text{insert}}(|s|, i)$ on the total time needed by a call of *insert*(s, i) and a time bound $t^*_{\text{insert}}(|s|, i)$ where we *do not count the time needed by the recursive calls* (made to *insert* in line 26), that is, for a "stripped" version of the

method where no recursive calls are made. We can later account for the missing calls by summing up over all calls that could possibly be made (but we count each only once, as we just observed that subsequent calls for the same parameters return immediately). In a similar fashion, let us try to establish time bounds $t_{\text{fix}}(|s'|, i)$ and $t^*_{\text{fix}}(|s'|, i)$ on the time needed (including or excluding the time needed by calls to *insert*) by a call to the method *fix needs downward*(s, s', i) (note that, indeed, these times are largely independent of s and its size – it is the size of s' that matters).

The starred versions are easy to bound: We have $t^*_{\text{insert}}(|s|, i) = O(1) + t^*_{\text{fix}}(|s|, i)$ as we call *fix needs downward* for $s' = s$. We have $t^*_{\text{fix}}(|s'|, i) = 2^{|s'|} \operatorname{poly} |s'|$ since the run-time is clearly dominated by the loop in line 18, which iterates over all subsets s'' of s'. For each of these $2^{|s'|}$ many sets, we run a test in line 20 that needs time $O(|s'|)$, yielding a total run-time of $t^*_{\text{fix}}(|s'|, i) = O(|s'|2^{|s'|})$. For the unstarred version we get:

$$\begin{aligned} t_{\text{insert}}(|s|, i) &= t^*_{\text{insert}}(|s|, i) + \textstyle\sum_{s' \subsetneq s, j \in \{|s'|, \ldots, i-1\}} t^*_{\text{insert}}(|s'|, j) \\ &= t^*_{\text{insert}}(|s|, i) + \textstyle\sum_{c=0}^{|s|-1} \underbrace{\binom{|s|}{c}}_{\text{number of } s' \subseteq s \text{ with } |s'|=c} \sum_{j=c}^{i-1} t^*_{\text{insert}}(c, j) \end{aligned}$$

Pluggin in the bound $2^c \operatorname{poly}(c)$ for $t^*_{\text{insert}}(c, j)$, we get that everything following the binomial can be bounded by $(d-c)2^c \operatorname{poly}(c) = 2^c \operatorname{poly}'(c)$. This means that the main sum we need to bound is $\sum_{c=0}^{|s|-1} \binom{|s|}{c} 2^c \leq \sum_{c=0}^{|s|} \binom{|s|}{c} 2^c$. The latter is equal to $3^{|s|}$, which yields the claim. ◀

▶ **Lemma 23.** DynKernelHS.*delete*(e) *runs in time* $5^d \operatorname{poly}(d)$.

The proof, to be found in the technical report version, is similar to the insertion case, but with some complications resulting from the fact that deletions may trigger insertions.

Proof of Theorem 1. The claim follows from Lemmas 18, 21, 22, and 23. ◀

5 Dynamic Set Packing Kernels

Like the static kernel [1], the dynamic kernel algorithm we have developed in the previous section also works, after a slight modification, for the set packing problem, which is the "dual" of the hitting set problem: Instead of trying to "cover" *all* hyperedges using as few vertices as possible, we must now "pack" *as many* hyperedges as possible. These superficially quite different problems allow similar kernel algorithms because correctness of the dynamic hitting set kernel algorithm hinges on Lemma 11, which states that every size-k hitting set X must hit the core of any b-flower F with $|F| > b \cdot k$. It leads to the central idea behind the complex management of the lists $L^i[s]$: The lists $L^i[s]$ were all b-flowers for different values of b by construction and the moment one of them gets larger than $b \cdot k$, we stop adding hyperedges to its relevant part and instead "switch over to the core s" by adding s to $L^{i-1}[s]$. It turns out that a similar lemma also holds for set packings:

▶ **Lemma 24.** *Let F be a b-flower with core c in a d-hypergraph $H = (V, E)$ and let $|F| > b \cdot d \cdot (k-1)$. If $E \cup \{c\}$ has a packing of size k, so does E.*

Proof. Let P be the size-k packing of $E \cup \{c\}$. If $c \notin P$, we are done, so assume $c \in P$. For each $p \in P - \{c\}$, consider the hyperedges in $e \in F$ with $p \cap e \neq \emptyset$. Since p has at most d elements v and since each v lies in at most b different hyperedges of the b-flower F, we conclude that p intersects with at most $d \cdot b$ hyperedges in F. However, this means that the $(k-1)$ different $p \notin P - \{c\}$ can intersect with at most $(k-1) \cdot b \cdot d$ hyperedges in F. In particular, there is a hyperedge $f \in F$ with $f \cap p = \emptyset$ for all $p \in P - \{c\}$. Since $F \subseteq E$, we get that $P - \{c\} \cup \{f\}$ is a packing of E of size k. ◀

Keeping this lemma in mind, suppose we modify the relevance bounds of the lists $L^i[s]$ as follows: Instead of setting them to $k^{i-|s|}$, we set them to $(d(k-1))^{i-|s|}$. Then all lists are b-flowers for a value of b such that whenever more than $b \cdot d(k-1)$ hyperedges are in $L^i[s]$, the set s gets forced into $L^{i-1}[s]$. Lemma 24 now essentially tells us that instead of considering the flower $E(L^i[s])$, it suffices to consider the core s (see also [3, Theorem 3] for more details). Thus, simply by replacing line 5 inside the *init* method as follows, we get a dynamic kernel algorithm for p_k-d-SET-PACKING:

5 (***new*** RELEVANCE LIST$\big(\big(d(k-1)\big)^{i-|s|}\big)$) ***for*** $s \in \binom{V}{\leq i}$ // Modified relevance bounds

6 Conclusion

We have introduced a fully dynamic algorithm that maintains a p_k-d-HITTING-SET kernel of size $\sum_{i=0}^{d} k^i \leq (k+1)^d$ with update time $5^d \operatorname{poly}(d)$ – which is a *constant, deterministic, worst-case* bound – and zero query time. Since p_k-d-HITTING-SET has no kernel of size $O(k^{d-\epsilon})$ unless coNP $\subseteq$ NP/poly [13], and since the currently best static algorithm requires time $|E| \cdot 2^d \operatorname{poly}(d)$ [30], this paper essentially settles the dynamic complexity of computing hitting set kernels. While it seems possible that the update time can be bounded even tighter with an amortized analysis, we remark that this could, at best, yield an improvement from the *already constant* worst-case time $5^d \operatorname{poly}(d)$ to an amortized time of $2^d \operatorname{poly}(d)$.

Our algorithm has the useful property that any size-k hitting set of a kernel is a size-k hitting set of the input graph. Therefore, we can also dynamically provide the following "gap" approximation with constant query time: Given a dynamic hypergraph H and a number k, at any time the algorithm either correctly concludes that there is no size-k hitting set, or provides a hitting set of size at most $\sum_{i=0}^{d} k^i$. With a query time that is linear with respect to the kernel size, we can also greedily obtain a solution of size dk, which gives a simple d-approximation. A "real" dynamic approximation algorithm, however, should combine the concept of α-approximate pre-processing algorithms [18, 26] with dynamic updates of the hypergraph. This seems manageable if we allow only edge insertions, but a solution for the general case is not obvious to us.

References

1 F. N. Abu-Khzam. A Kernelization Algorithm for d-Hitting Set. *Journal of Computer and System Sciences*, 76(7):524–531, 2010. doi:10.1016/j.jcss.2009.09.002.

2 Josh Alman, Matthias Mnich, and Virginia Vassilevska Williams. Dynamic parameterized problems and algorithms. *ACM Trans. Algorithms*, 16(4):1–46, July 2020. doi:10.1145/3395037.

3 Max Bannach, Zacharias Heinrich, Rüdiger Reischuk, and Till Tantau. Dynamic kernels for hitting sets and set packing. Technical Report TR19-146, Computational Complexity Foundation, 2019. URL: https://eccc.weizmann.ac.il/report/2019/146.

4 Max Bannach, Malte Skambath, and Till Tantau. Kernelizing the hitting set problem in linear sequential and constant parallel time. In *17th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2020, June 22–24, 2020, Tórshavn, Faroe Islands*, pages 9:1–9:16, 2020. doi:10.4230/LIPIcs.SWAT.2020.9.

5 Max Bannach and Till Tantau. Computing Hitting Set Kernels By AC^0-Circuits. *Theory Comput. Syst.*, 64(3):374–399, 2020. doi:10.1007/s00224-019-09941-z.

6 S. Bhattacharya, M. Henzinger, and G. F. Italiano. Deterministic Fully Dynamic Data Structures for Vertex Cover and Matching. In *Proceedings of the 26th ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4–6, 2015*, pages 785–804, 2015. doi:10.1137/1.9781611973730.54.

7 J. F. Buss and J. Goldsmith. Nondeterminism Within P. *SIAM Journal on Computing*, 22(3):560–572, 1993. doi:10.1137/0222038.

8 Jiehua Chen, Wojciech Czerwinski, Yann Disser, Andreas Emil Feldmann, Danny Hermelin, Wojciech Nadara, Marcin Pilipczuk, Michal Pilipczuk, Manuel Sorge, Bartlomiej Wróblewski, and Anna Zych-Pawlewicz. Efficient fully dynamic elimination forests with applications to detecting long paths and cycles. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10–13, 2021*, pages 796–809. SIAM, 2021. doi:10.1137/1.9781611976465.50.

9 Y. Chen, J. Flum, and X. Huang. Slicewise Definability in First-Order Logic with Bounded Quantifier Rank. In *Proceedings of the 26th EACSL Annual Conference on Computer Science Logic, CSL 2017, August 20–24, 2017, Stockholm, Sweden*, pages 19:1–19:16, 2017. doi:10.4230/LIPIcs.CSL.2017.19.

10 M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer Berlin Heidelberg, 2015.

11 P. Damaschke. Parameterized Enumeration, Transversals, and Imperfect Phylogeny Reconstruction. *Theoretical Computer Science*, 351(3):337–350, 2006. doi:10.1016/j.tcs.2005.10.004.

12 S. Datta, R. Kulkarni, A. Mukherjee, T. Schwentick, and T. Zeume. Reachability Is in DynFO. *Journal of the ACM*, 65(5):33:1–33:24, 2018. doi:10.1145/3212685.

13 H. Dell and D. 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.

14 R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

15 Rodney G. Downey, Michael R. Fellows, and Ulrike Stege. Parameterized complexity: A framework for systematically confronting computational intractability. In Ronald L. Graham, Jan Kratochvíl, Jaroslav Nesetril, and Fred S. Roberts, editors, *Contemporary Trends in Discrete Mathematics: From DIMACS and DIMATIA to the Future, Proceedings of a DIMACS Workshop, Stirín Castle, Czech Republic, May 19–25, 1997*, volume 49 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 49–99. DIMACS/AMS, 1997. doi:10.1090/dimacs/049/04.

16 P. Erdős and R.Rado. Intersection Theorems for Systems of Sets. *Journal of the London Mathematical Society*, 1(1):85–90, 1960.

17 Stefan Fafianie and Stefan Kratsch. A shortcut to (sun)flowers: Kernels in logarithmic space or linear time. In *Proceedings of the 40th International Symposium on Mathematical Foundations of Computer Science, MFCS 2015, Milan, Italy, August 24–28, 2015*, volume 9235 of *Lecture Notes in Computer Science*, pages 299–310. Springer, 2015. doi:10.1007/978-3-662-48054-0_25.

18 Michael R. Fellows, Ariel Kulik, Frances A. Rosamond, and Hadas Shachnai. Parameterized approximation via fidelity preserving transformations. *J. Comput. Syst. Sci.*, 93:30–40, 2018. doi:10.1016/j.jcss.2017.11.001.

19 Henning Fernau. A top-down approach to search-trees: Improved algorithmics for 3-hitting set. *Algorithmica*, 57(1):97–118, 2010. doi:10.1007/s00453-008-9199-6.

20 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. Springer, 2006. doi:10.1007/3-540-29953-X.

21 Fedor V. Fomin, Serge Gaspers, Dieter Kratsch, Mathieu Liedloff, and Saket Saurabh. Iterative compression and exact algorithms. *Theor. Comput. Sci.*, 411(7–9):1045–1053, 2010. doi:10.1016/j.tcs.2009.11.012.

22 M. Henzinger and V. King. Maintaining Minimum Spanning Forests in Dynamic Graphs. *SIAM Journal on Computing*, 31(2):364–374, 2001. doi:10.1137/S0097539797327209.

23 J. Holm, K. de Lichtenberg, and M. Thorup. Poly-Logarithmic Deterministic Fully-Dynamic Algorithms for Connectivity, Minimum Spanning Tree, 2-Edge, and Biconnectivity. *Journal of the ACM*, 48(4):723–760, 2001. doi:10.1145/502090.502095.

24 Y. Iwata and K. Oka. Fast Dynamic Graph Algorithms for Parameterized Problems. In *Proceedings of the 14th Scandinavian Symposium and Workshop on Algorithm Theory, SWAT 2014, Copenhagen, Denmark, July 2–4, 2014*, pages 241–252, 2014. doi:10.1007/978-3-319-08404-6_21.

25 R. M. Karp. Reducibility Among Combinatorial Problems. In *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*, pages 85–103, 1972. doi:10.1007/978-1-4684-2001-2_9.

26 Daniel Lokshtanov, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. Lossy kernelization. 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 224–237. ACM, 2017. doi:10.1145/3055399.3055456.

27 Kurt Mehlhorn. *Data Structures and Algorithms 1: Sorting and Searching*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1984.

28 R. Niedermeier and P. 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.

29 S. Patnaik and N. Immerman. DynFO: A Parallel, Dynamic Complexity Class. *Journal of Computer and System Sciences*, 55(2):199–209, 1997. doi:10.1006/jcss.1997.1520.

30 R. van Bevern. Towards Optimal and Expressive Kernelization for d-Hitting Set. *Algorithmica*, 70(1):129–147, September 2014. doi:10.1007/s00453-013-9774-3.

31 René van Bevern. *Fixed-Parameter Linear-Time Algorithms for NP-hard Graph and Hypergraph Problems Arising in Industrial Applications*, volume 1 of *Foundations of Computing*. Universitätsverlag der TU Berlin, 2014. doi:10.14279/depositonce-4131.

32 René van Bevern and Pavel V. Smirnov. Optimal-size problem kernels for d-hitting set in linear time and space. *Information Processing Letters*, 163(105998), 2020. doi:10.1016/j.ipl.2020.105998.

33 Magnus Wahlström. *Algorithms, measures and upper bounds for satisfiability and related problems*. PhD thesis, Linköping University, Sweden, 2007.

(Sub)linear Kernels for Edge Modification Problems Towards Structured Graph Classes

Gabriel Bathie
École Normale Supérieure de Lyon, France

Nicolas Bousquet
LIRIS, CNRS, Université Claude Bernard Lyon 1, Université de Lyon, France

Théo Pierron
LIRIS, CNRS, Université Claude Bernard Lyon 1, Université de Lyon, France

Abstract

In a (parameterized) graph edge modification problem, we are given a graph G, an integer k and a (usually well-structured) class of graphs $\mathcal{G}$, and ask whether it is possible to transform G into a graph $G' \in \mathcal{G}$ by adding and/or removing at most k edges. Parameterized graph edge modification problems received considerable attention in the last decades.

In this paper, we focus on finding small kernels for edge modification problems. One of the most studied problems is the Cluster Editing problem, in which the goal is to partition the vertex set into a disjoint union of cliques. Even if this problem admits a $2k$ kernel [7], this kernel does not reduce the size of most instances. Therefore, we explore the question of whether linear kernels are a theoretical limit in edge modification problems, in particular when the target graphs are very structured (such as a partition into cliques for instance). We prove, as far as we know, the first sublinear kernel for an edge modification problem. Namely, we show that Clique + Independent Set Deletion, which is a restriction of Cluster Deletion, admits a kernel of size $O(k/\log k)$.

We also obtain small kernels for several other edge modification problems. We prove that Split Addition (and the equivalent Split Deletion) admits a linear kernel, improving the existing quadratic kernel of Ghosh et al. [19]. We complement this result by proving that Trivially Perfect Addition admits a quadratic kernel (improving the cubic kernel of Guo [21]), and finally prove that its triangle-free version (Starforest Deletion) admits a linear kernel, which is optimal under ETH.

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

Keywords and phrases kernelization, graph editing, split graphs, (sub)linear kernels

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.8

Related Version *Full Version*: `https://arxiv.org/abs/2105.09566`

Funding This work was supported by ANR project GrR (ANR-18-CE40-0032).

1 Introduction

A central problem in the context of data transmission, collection or storage, is to recover the original information when the data has been altered. Although it is not possible to know what the original data was in the general setting, it may be possible when we have some knowledge of the structure of the original data. When we know that the alteration is limited, it is reasonable to assume that the original data is an element that has the desired structure and is the closest to the altered data.

When the data that we are reconstructing is a graph, the problem becomes the following: given a graph G (the altered data) and a class of graphs $\mathcal{G}$ (the structure of the data), find the graph in $\mathcal{G}$ that is the "closest" to G (candidate for the original data). There are multiple

ways to define the distance between two graphs, but the most widely used is the minimum number of vertex modifications or edge modifications needed to turn one into the other. This type of problem, called graph modification problems, received considerable attention, for instance in computational biology [2], machine learning [1], and image processing [28].

In this work, we focus on edge modification problems, i.e. the distance is the minimum number of edge modifications (see Section 2 for formal definitions of these problems). A line of work, initiated by Yannakakis [29], showed that deciding whether a graph G is at distance at most k from $\mathcal{G}$ is NP-complete for most classes of graphs, even for very restricted classes such as bipartite graphs. See [5, 25, 26] for an overview of the different results.

Therefore, in the last decades, edge modification problems received considerable attention from the point of view of parameterized complexity, which studies the resources required to solve NP-complete problems in a fine-grained way. See for example [3, 4, 6, 12, 15, 19, 27], and see [10] for a recent survey on the topic. In this paper, we will consider problems parameterized by the size k of the solution (that is, the set of edges to add or remove).

In this work, we focus on graph classes that can be characterized by a finite number of forbidden induced subgraphs. In his seminal paper, Cai [6] showed that, for every class of graphs $\mathcal{G}$ that can be characterized by a finite number of forbidden induced subgraphs, the $\mathcal{G}$-edge modifications problems are FPT parameterized by k. In other words, there exists a constant c, a function f (that only depend on $\mathcal{G}$), and an algorithm running in time $f(k) \cdot n^c$ that either finds a solution of size at most k, or returns that there is no such solution. Therefore, most of the subsequent efforts focused on determining for which $\mathcal{G}$ these problems admit a polynomial kernel. Intuitively, a kernel is a polynomial-time preprocessing algorithm that extracts the "hard" part of an instance (G, k): it solves easy parts of the instance and returns an equivalent instance (G', k'), whose size is bounded by $f(k)$, for some function f. A kernel is a polynomial kernel if f is a polynomial. The interested reader is referred to [18] for more details.

One of the most studied edge modification problem is Cluster Editing, in which the goal is to partition the graph into a disjoint union of cliques. This problem is known to admit a kernel with at most $2k$ vertices. While this result seems impressive at first glance (for many parameterized problems, a linear kernel is asymptotically optimal), we can remark that here, we are comparing the number of vertices of the kernel with the number of edges in the solution. It turns out that for most graphs in practice, the number of edges that have to be modified to obtain a cluster graph is larger than the number of vertices. For example, it is the case for most of the public instances of the PACE challenge 2021 on cluster editing[1].

This raises the question of whether linear kernels are optimal, in particular for Cluster Editing. We partially answer this question by giving a sublinear kernel for the closely related Clique + IS Deletion problem. It provides a "proof of concept" that linear kernels are not always optimal. As far as we know, it is the first example of a sublinear kernel for a graph edge modification problem. We complete this result with linear or quadratic kernels for several other edge modification problems.

Due to space constraints, all the proofs that are not in this extended abstract can be found in the appendix.

Our results. In this work, our goal is to understand when it is possible to obtain small, and in particular linear or sublinear kernels for edge modification problems. We focus in particular on graph classes where the vertex set can be partitioned into highly structured classes such as cliques or independent sets. A typical example of such graph class is the class of split graphs, i.e. graphs that can be partitioned into a clique and an independent set.

[1] See `https://pacechallenge.org/2021/` for more information.

Most of our results are based on a high-level technique, that we call Label-And-Reduce, which helps to design efficient kernelization algorithms for edge modification problems. The key idea is to use the strong structure of each of the graphs in $\mathcal{G}$ to find a highly structured partition $X_1, \dots, X_\ell$ of the graphs of $\mathcal{G}$ (e.g. a partition in cliques or independent sets, complete bipartition between subsets...). We then define rules that label vertices x as belonging to X_i, in such a way that if there is a solution, there is one for which $x \in X_i$. We finally show that 1) when no rule can be applied, the number of unlabeled vertices is $O(\text{poly}(k))$ when (G, k) is a positive instance and 2) the number of labeled vertices in each class X_i can be reduced to $O(\text{poly}(k))$.

Sublinear kernel for Clique+Independent Set deletion. The problems of graph edge modification towards cluster graphs have received considerable attention in the last two decades in parameterized complexity, see e.g. [4, 7, 9, 17, 27].

The deletion version, Cluster Deletion admits a cubic kernel [20]. We focus on a restricted version of this problem, where all clusters but at most one have size 1. It corresponds to graphs that are the disjoint union of a clique and an independent set. In what follows, we will refer to this class as the class of *clique + IS graphs*, and to the corresponding problem as Clique + IS Deletion. Since clique + IS graphs are $(P_3, 2K_2)$-free graphs, Clique + IS Deletion is FPT by [6].

While Clique + IS Addition is trivial, both Clique + IS Deletion and Clique + IS Edition are NP-complete (reduction from the Clique problem), and both can be solved in subexponential time ($O^*(1.64^{\sqrt{k \ln k}})$ and $O^*(2^{\sqrt{k \ln k}})$ respectively[2]) [12]. Both problems also admit a simple $2k$-kernel, based on twin reduction rules [10].

Our result, proved in Section 3, is the following.

▶ **Theorem 1.1.** *Clique + IS Deletion admits a kernel of size* $2k/\log k + 1$.

Our algorithm uses the structure of clique+IS graphs to remove vertices with small degree and to reduce the instance when the minimum degree of the input is large. Theorem 1.1 is, as far as we know, the first sublinear kernel for edge modification problems. We conjecture that the size of this kernel is not optimal, and ask the following:

▶ **Open Problem 1.** Is there an $O(k^{1-\varepsilon})$ kernel for Clique + IS Deletion for some $\varepsilon > 0$?

Moreover, it is plausible that other edge modification problems towards highly structured classes also admit a sublinear kernel. A natural candidate is the closely related Cluster Editing problem, which already admits a $2k$ kernel [7, 9].

▶ **Open Problem 2.** Does Cluster Editing (resp. Cluster Deletion) admit a sublinear kernel?

Linear kernel for Split addition. Split graphs are graphs whose vertex set can be partitioned into a clique K and an independent set I (with no constraint on the set of edges between K and I). Since split graphs are auto-complementary, the Split Addition and Split Deletion problems are equivalent. Natanzon et al. showed that these two problems are NP-complete [26]. Since split graphs are $(2K_2, P_4, C_5)$-free graphs, the latter problems are

[2] Recall that O^* denotes the complexity up to polynomial factors.

FPT [6]. Ghosh et al. [19] later showed that these problems can be solved in subexponential $O^*(2^{O(\sqrt{k}\log k)})$ time and that they admit a quadratic kernel. Cygan et al. [11] improved the complexity to $O^*(2^{\sqrt{k}})$. Hammer and Simeone [22] showed that, rather surprisingly, the related Split Edition problem can be decided in polynomial time.

We improve upon the result of Ghosh et al. [19] by showing that the Split Addition (and therefore Split Deletion) problem admits a linear kernel in Section 4.

▶ **Theorem 1.2.** *Split Addition and Split Deletion admit a kernel with at most* $11k + 6\sqrt{2k} + 4$ *vertices.*

This result is the main technical contribution of the paper. From a very high-level perspective, our algorithm works as follows. Let (G, k) be a positive instance. If the clique of the solution is large enough, the neighborhood of many vertices of that clique has not been modified and we show that we can detect some of them and label them as clique vertices. Since the number of unlabeled clique vertices of the solution is bounded by a linear function, we can prove via a tricky and short argument that the number of unlabeled vertices of the independent set can be bounded. While the reduction rules are not very complicated, showing that the answer is negative when the number of unlabeled vertices is too large is the core of the proof. We finally show that we reduce the number of labeled vertices to $O(k)$ vertices.

Quadratic kernel for trivially perfect graphs. A trivially perfect graph is a graph such that for any pair of adjacent vertices u, v, $N(u) \subseteq N(v)$ or $N(v) \subseteq N(u)$. The class of trivially perfect graphs can equivalently be characterized as the class of (P_4, C_4)-free graphs. Drange et al. [13, 14] showed that, under the Exponential Time Hypothesis (ETH), Trivially Perfect Deletion and Trivially Perfect Edition cannot be solved in subexponential time. Liu et al. [24] gave an FPT algorithm for Trivially Perfect Deletion running in time $O^*(2.42^k)$. On the other hand, the Trivially Perfect Addition problem does not admit such lower bounds. Drange et al. [13] designed a subexponential $O(2^{\sqrt{k \log k}}))$ algorithm for the problem and Bliznets et al. [3] showed that assuming ETH, this cannot be improved beyond $O(2^{k^{1/4}})$. In 2018, Drange and Pilipczuk [14] showed that the three problems admit a polynomial kernel of size $O(k^7)$, recently improved by Dumas et al. [16] into $O(k^3)$.

In the specific case of Trivially Perfect Addition, a cubic kernel was already provided by Guo [21]. We improve this result in Section 5 by showing the following.

▶ **Theorem 1.3.** *Trivially Perfect Addition admits a kernel with* $2k^2 + 2k$ *vertices.*

Our kernel is based on the claim of Guo [21], which states that the instance can be reduced to vertices that belong to at least one obstruction (that is, an induced P_4 or C_4). Using this claim, Guo proved the existence of a cubic kernel. By counting obstructions more precisely, we actually show very simply that the size of the kernel can be reduced to $O(k^2)$.

Linear kernelization of starforests. We finally focus on triangle-free trivially perfect graphs, also known as star forests. A *star* is a tree with at most one internal vertex. A star with n vertices is called an *n-star*. Note that the single vertex graph and K_2 are stars. The class of *starforests* graphs is the class of graphs that are a disjoint union of stars, that is every connected component is a star. Alternatively, it may be defined as the class of K_3, C_4, P_4-free graphs.

One can remark that removing an edge from a starforest yields another starforest, hence it is never interesting to add edges to obtain a star forest. Therefore, Starforest Addition is trivial, and Starforest Edition is equivalent to Starforest Deletion. Drange et al. showed in [15] that Starforest Deletion is NP-complete and cannot be solved in subexponential time (that is in time $O(2^{o(k)}\operatorname{poly}(n))$), assuming the ETH [23].

In Section 6, we prove the following result.

▶ **Theorem 1.4.** *Starforest Deletion admits a kernel with at most $4k+2$ vertices.*

We also show that, under ETH, Starforest Deletion does not admit a sublinear kernel. To the best of our knowledge, this work is the first formally published work on kernelization of Starforest Deletion.

Note. Cao and Yuping [8] obtained independently at the same time results that are very similar to ours: they designed the same quadratic kernel for Trivially Perfect Addition and obtained a similar kernel for Split Addition. However, they were only able to prove an $O(k^{1.5})$ upper bound for the latter, whereas we prove a tighter $O(k)$ bound.

2 Preliminaries

Elementary definitions. In this work, all the graphs are undirected and simple (i.e. with no parallel edges or self-loops). When G is a graph, $V(G)$ denotes the set of vertices of G, and $E(G)$ denotes its set of edges. Throughout the paper, we use n (resp. m) to denote the size of $V(G)$ (resp. $E(G)$). If $uv \in E(G)$, we say that u and v are *adjacent*. Given a vertex $u \in V(G)$, $N(u) = \{v \text{ such that } uv \text{ is an edge}\}$ is the *open neighborhood* of u, and $N[u] = N(u) \cup \{u\}$ is the *closed neighborhood* of u. The *degree* of u in G, denoted $d(u)$, is the size of $N(u)$. We use $\delta(G)$ to denote the minimum degree of G. The *complement graph* $\bar{G}$ of G is the graph with vertex set $V(G)$ and edge set $\{uv : u \neq v \text{ and } uv \notin E(G)\}$. A *dominating set* of G is a set D of vertices of G such that every vertex of G is either in D or adjacent to a vertex of D. An *independent set* of G is a set I of vertices of G that are pairwise not adjacent.

Kernelization algorithms. A *kernelization algorithm* (in short, a kernel) is a polynomial-time algorithm that takes as input an instance (G,k) of a parameterized problem Π and outputs an instance (G',k') that is positive if and only if (G,k) is positive, the size of G' is at most $f(k')$ for some computable function f. When f is a polynomial, we say that the algorithm is a *polynomial kernel*. When dealing with graph problems, the size of the instance is often measured in terms of the number of vertices of G'. Most kernelization algorithms (including those presented in this report) consist of the iterative application of *reduction rules*. A reduction rule is a polynomial-time algorithm that input an instance (G,k) and outputs another instance (G',k'). We say that a reduction rule R is *safe* when (G',k') is positive if and only if (G,k) is.

Graph edge modification problems. Let $\mathcal{G}$ be a class of graphs. In a (parameterized) $\mathcal{G}$-graph edge modification problem, we are given a graph G, an integer k, and ask whether it is possible to transform G into a graph $G' \in \mathcal{G}$ by modifying (*adding*, *removing*, or doing both, which is called *editing*) at most k edges.

Given a set of edges F, we use the notation $G+F$, $G-F$, and $G\Delta F$ to denote the graphs with vertex set $V(G)$ and respective set of edges $E(G) \cup F$, $E(G) \setminus F$ and $E(G)\Delta F$.

Formally, we will consider the following problems:

▶ **Problem 1** ($\mathcal{G}$-Addition (resp. Deletion, resp. Edition)).
Input: A graph G, an integer $k \in \mathbb{N}$.
Output: "YES" if there exists a set F of at most k edges of G such that $G+F$ (resp. $G-F$, resp. $G\Delta F$) is in $\mathcal{G}$, "NO" otherwise.

3 Sublinear kernel for the Clique + Independent set deletion problem

The goal of this section is to prove Theorem 1.1. To obtain the announced kernel, we apply the Label-And-Reduce technique. For this problem, the labeling rules aim to identify vertices that will be in the independent set of a solution if it exists. We can then delete all the edges incident to these vertices, decrease the parameter accordingly and remove these vertices from the graph.

We assume that k is smaller than m since otherwise, the instance is trivial: we obtain an independent set (which is a clique+IS graph) by deleting all the edges in G.

▶ **Rule 3.1** (Low degree reduction rule 1). *If $v \in V(G)$ has degree $d(v) < \sqrt{2(m-k)} - 1$, delete v from G and decrease the parameter by $d(v)$.*

This rule can be implemented to run in linear time. It is moreover safe. Indeed, since we consider the deletion problem, any vertex v deleted by the rule has degree smaller than $\sqrt{2(m-k)} - 1$ in $G - F$, hence cannot be in the clique of an optimal solution according to the following lemma.

▶ **Lemma 3.2.** *Let (G, k) is a positive instance of Clique + IS Deletion. If F is a solution of (G, k), then the clique in $G - F$ has size at least $\sqrt{2(m-k)}$.*

Proof. Since F contains at most k edges, the graph $G-F$ has at least $m-k$ edges. Moreover, $G-F$ is a clique+IS graph, therefore all its edges are the edges of a unique clique. Therefore, the size c of the clique of $G-F$ satisfies $\binom{c}{2} \geqslant m-k$, hence $c \geqslant \sqrt{2(m-k)}$. ◀

One can prove that this first rule can be extended to obtain a linear kernel: when this rule cannot be applied, assuming that $m \geq 2k$ implies that $n = O(\sqrt{m})$. When $m = O(k^2)$, we are done, and when $m = \Omega(k^2)$, the minimum degree of the graph is $\Omega(k)$, therefore at most $O(1)$ vertices can be removed, and in that case, the existence of a solution can be tested in polynomial time.

To further reduce the size of the kernel to $O(k/\log k)$, we use the two following rules to take care of very sparse or very dense instances.

▶ **Rule 3.3** (Low degree reduction rule 2). *Let v be a vertex of degree at most $2\log k - 1$. If there is no solution F of (G, k) such that v is in the clique of $G - F$, remove v from G and decrease k by $d(v)$.*

This rule is trivially safe. Moreover, it can be performed in polynomial time. Indeed, since we consider an edge-deletion problem, if v lies in the clique K of $G - F$, then every vertex of K is adjacent to v in G, i.e. $K \subseteq N[v]$. Since the degree of v is at most $2\log k - 1$, there are at most k^2 subsets in $N[v]$. We can therefore try all of them and decide in polynomial time whether there exists a solution F of (G, k) such that v is in the clique in $G - F$.

▶ **Rule 3.4** (High degree). *If G has minimum degree $\delta(G) \geq k/(2\log k)$, solve the instance and output a trivial equivalent instance.*

Again, this rule is clearly safe. The not-so-easy part is to show that Clique + IS Deletion can be decided in polynomial time when $\delta(G) \geq k/(2\log k)$.

▶ **Lemma 3.5.** *Rule 3.4 can be applied in polynomial time.*

To finish the proof of Theorem 1.1, it remains to bound the size of the reduced graph. This is the goal of the following lemma.

▶ **Lemma 3.6.** *If (G, k) is a positive instance and none of the rules can be applied, then $|V(G)| \leq 2 \cdot \frac{k}{\log k} + 1$.*

Concluding remarks. Finally, one can easily show that Rule 3.1 can be adapted for the Clique + IS Edition problem by modifying the constant. On the other hand, it seems that Rules 3.3 and 3.4 do not readily generalize to Clique + IS Edition, therefore we were not able to obtain an $O(k/\log k)$ kernel for this problem. However, it is an easy exercise to show that we can weaken them to obtain a kernel with at most k/c vertices for any possible constant $c > 1$, at the cost of a running time in $O(n^c)$.

4 Linear kernel for addition towards split graphs

The goal of this section is to prove Theorem 1.2. Since the class of split graphs is closed under complementation, it is sufficient to prove that Theorem 1.2 holds for Split Addition.

We use the structure of the input graph to detect and label vertices that will be in the clique or the independent set part of a split decomposition of a well-chosen solution. More precisely, we show that, if the instance (G, k) is positive, the labeling constructed by our algorithm satisfies that there exists a solution F of (G, k) and a split decomposition (K^*, I^*) of $G + F$ such that all the vertices labeled as "clique" (resp. "independent set") are in K^* (resp. I^*). We then prove that if (G, k) is a positive instance, then the number of unlabeled vertices at the end of the algorithm is $O(k)$. Moreover, we show that we can reduce the number of labeled vertices to $O(k)$. Combining the above yields a linear kernel.

We present our reduction rules in Section 4.1 and prove them in subsequent sections.

4.1 Labeling and reduction rules

Our algorithm keeps track of a partition (K, I, D) of $V(G)$, which corresponds to the labels of the vertices of G. The set K (resp. I) stands for the vertices already labeled "clique" (resp. "independent set") while D (for "don't know") contains the vertices that are not yet labeled. Initially, no vertex is labeled, hence $K = \emptyset, I = \emptyset$ and $D = V(G)$.

We will apply the following reduction rules, whose correction is postponed to Section 4.2.

▶ **Rule 4.1** (I-rules). *Move $v \in D$ to I whenever at least one of the following holds:*
(a) *v has all of its neighbors in K,*
(b) *v is non-adjacent to at least $k + 1$ vertices of K.*
Notice that this rule applies to isolated vertices since whenever v is isolated, $N(v) = \emptyset \subseteq K$.

▶ **Rule 4.2** (K-rules). *Move $v \in D$ to K whenever at least one of the following holds:*
(a) *v has a neighbor in I,*
(b) *$N(v)$ contains at least $k + 1$ non-edges,*
(c) *v dominates $K \cup D$.*

The following reduction rule simply ensures that K is a clique and I an independent set.

▶ **Rule 4.3** (Reduction rules). *Apply one of the following rules as long as possible:*
(a) *if there is a non-edge e between vertices of K, then add e to $E(G)$ and decrease k by 1.*
(b) *if $k < 0$ then return a trivially negative instance.*
(c) *if there is an edge between vertices of I, then return a trivially negative instance.*

We apply these rules exhaustively, and stop when none can be applied. At each step, we remove a vertex from D or we add an edge to G. Then the algorithm stops after at most n^2 steps. Moreover, one can easily apply the rules in polynomial time. When none of the previous rules can be applied, we apply the following reduction rule.

▶ **Rule 4.4** (Unlabeling algorithm).

(a) *If K contains at least $k+1$ vertices, replace K by a set $K' = \{v'_1, \ldots, v'_k\}$ of k vertices and denote by G' the resulting graph. Moreover, for each vertex $v \in D$, if v is non-adjacent to t vertices of K in G, then connect v to $v'_{t+1}, \ldots, v'_k$ and do not connect it to $v'_1, \ldots, v'_t$.*

(b) *Replace I by an independent set I' of size $\sqrt{2k}$ connected to K' in a complete bipartite manner, and not connected to D.*

With this rule, we can bound the number of vertices of the resulting graph by $|D|$ plus at most k vertices (for K'), plus at most $\sqrt{2k}$ vertices (for I'). Therefore, Theorem 1.2 boils down to the following lemma.

▶ **Lemma 4.5.** *If (G, k) is a positive instance, then $|D| \leq 10k + 5\sqrt{2k} + 4$.*

While it is not very difficult to prove that the reduction rules are safe, the main technical contribution of this section consists in proving Lemma 4.5. The proof is split into two parts. First, we prove that the number of vertices of D in the clique K^* of the solution is linear in k. We prove it by showing that, if too many vertices of K^* are in D, the neighborhood of many of them is not modified. And amongst them, one must be complete to $K \cup D$, a contradiction with Rule 4.2.

Arguing that the number of vertices of D in the independent set I^* of the solution is $O(k)$ is more involved. First note that if a vertex has an independent set of size larger than $O(\sqrt{k})$ in its neighborhood, it is added to K by Rule 4.2-b. Since D only contains $O(k)$ vertices in the clique, the number of vertices of D in the independent set is at most $O(k^{3/2})$. To obtain a better upper bound on the size of D, we carefully distinguish the size of the neighborhood of the vertices of $D \cap K^*$ in I^*. Very roughly, we prove that the number of vertices in $D \cap K^*$ with many neighbors in I^* is bounded by a sublinear function which permits to improve the size of the kernel. The proof of Lemma 4.5 is postponed to Section 4.3.

Lemma 4.5 together with Rule 4.4 ensure that the following reduction rule is correct, which completes the proof of Theorem 1.2:

▶ **Rule 4.6** (Final Rule). *If none of the previous rules can be applied, and the size of the instance is at least $11k + 6\sqrt{2k} + 5$, return a trivially negative instance.*

4.2 Correctness of the reduction rules

To analyze our algorithm, we study the evolution of the instance (G, k) with the partition $P = (K, I, D)$ after the application of each rule. We will refer to the tuple (G, k, P) as a *generalized instance* of Split Addition.

The following definition formalizes when a labeling of G is compatible with a solution F.

▶ **Definition 4.7.** *Let H be a graph, let F be a set of edges such that $H + F$ is a split graph, and let $P = (K, I, D)$ be a partition of $V(H)$. We say that P* is compatible with *F, and denote it $P \vDash F$, if there exists a split decomposition (K^*, I^*) of $H + F$ such that $K \subseteq K^*$ and $I \subseteq I^*$. In that case, we say that the decomposition (K^*, I^*)* witnesses *the fact $P \vDash F$.*

A generalized instance represents a graph along with a partial labeling of the vertices. Such an instance is positive when there exists a solution that is compatible with the labeling. This leads to the following definition.

▶ **Definition 4.8** (Positive generalized instance). *A generalized instance (G, k, P) is positive if there exists a solution F of (G, k) such that $P \vDash F$.*

This allows us to extend safeness properties to reduction rules operating on generalized instances. We now show that the labeling and reduction rules preserve the existence of a solution.

▶ **Lemma 4.9.** *Rules 4.1 to 4.3 are safe.*

Note that the initial labeling $P = (\emptyset, \emptyset, V(G))$ is compatible with every solution of (G, k) (if any). Therefore, by applying transitively Lemma 4.9, we get that the labeling and reduction process is safe.

We finally show that Rule 4.4 is safe.

▶ **Lemma 4.10.** *Rule 4.4 is safe.*

4.3 Structure of positive instances

This section is devoted to the proof of Lemma 4.5, restated below.

▶ **Lemma 4.5.** *If (G, k) is a positive instance, then $|D| \leq 10k + 5\sqrt{2k} + 4$.*

In what follows, we assume that the input is a positive instance and the labeling/reduction process stopped and returned a generalized instance (G, k, P). In particular, Rules 4.1 to 4.3 cannot be applied. By Lemma 4.9, we get that there exists a solution F of (G, k) such that $|F| \leq k$ and $P \vDash F$. Unrolling the definition, this means that there exists a split decomposition (K^*, I^*) of $G + F$ such that $K \subseteq K^*$ and $I \subseteq I^*$. Let $K^D = D \cap K^*$ be the set of unlabeled vertices that belong to the clique, and let $I^D = D \cap I^*$ be the set of the unlabeled vertices that belong to the independent set. For every $v \in D$, let $I_v = N(v) \cap I^D$. We give an upper bound on the cardinality of D by giving separate upper bounds on the respective cardinalities of K^D and I^D.

Before diving into the details of the proof, let us make two observations on the structure of D, that follow from the fact that the labeling rules cannot be applied.

▶ **Observation 4.11.** *For every vertex $v \in K^D$, $|I_v| \leq \sqrt{2k} + 1$.*

▶ **Observation 4.12.** *Every vertex $v \in I^D$ has a neighbor in K^D.*

We first prove that $|K^D| = O(k)$.

▶ **Lemma 4.13.** *We have $|K^D| \leq 4k$.*

Proof. Let us prove this statement by contradiction: we prove that if $|K^D| \geq 4k + 1$, then there is a vertex in D that dominates $D \cup K$, which contradicts the fact that Rule 4.2-c cannot be applied.

By assumption, K^* is a clique in $G + F$. Since F contains at most k edges, there are at most $2k$ vertices of K^D that are adjacent to edges of F. Since $|K^D| \geq 4k + 1$, there are at least $2k + 1$ vertices in K^D that dominate $K^* = K^D \cup K$. Let X denote the set of such vertices. We will now show that there is a vertex in X that also dominates I^D, that is, a vertex of K^D that dominates $K^D \cup I^D \cup K = D \cup K$. To prove the existence of this vertex, we will prove that for any vertex u in X such that $I_u \neq I^D$, there exists a vertex $v \in X$ such that $|I_v| > |I_u|$. By applying this property repeatedly, we eventually find a vertex v such that $I_v = I^D$.

Let u be a vertex of X such that $I_u \neq I^D$. Since $K^D \subseteq N[u]$ and Rule 4.2-b cannot be applied, there are at most k non-edges between I_u and K^D. Hence, these non-edges are adjacent to at most k vertices of X (X being a clique, every non-edge is incident to at most one vertex of X), and then at least $k+1$ vertices of X dominate I_u. Let X' be the subset of vertices of X that dominate $K^* \cup I_u$. Let w be a vertex of $I^D \setminus I_u$. As noted in Observation 4.12, w is adjacent to some vertex $v \in K^D$. Assume that w is anticomplete to X', so that $v \notin X'$. Since $v \in K^*$, every vertex of X' is adjacent to v. Therefore v contains at least $k+1$ non-edges in its neighborhood, namely the edges between w and X', a contradiction.

Therefore, $v \in X'$ and the conclusion follows since I_v contains I_u and w. ◀

By bounding locally the size of the neighborhood of each vertex in K^D using Observation 4.11, Lemma 4.13 directly provides an $O(k^{\frac{3}{2}})$ kernel. However, as we will show, this is not tight. Using a more global counting argument, we can show that $|I^D| = O(k)$.

▶ **Lemma 4.14.** *We have* $|I^D| \leq 6k + 5\sqrt{2k} + 4$.

Proof. First, notice that Observation 4.12 implies that $I^D \subseteq \bigcup_{v \in K^D} N(v)$. Therefore, if $|K^D| \leq \sqrt{8k}$, Observation 4.11 implies the following upper bound on the cardinality of I^D:

$$|I^D| \leq |K^D| \cdot (\sqrt{2k}+1) \leq 4k + 2\sqrt{2k} \leq 6k + 5\sqrt{2k} + 4.$$

In what follows, we assume that $|K^D| > \sqrt{8k}$. We partition I^D into two sets: I^+, the set of vertices that have degree at least $|K^D|/4$, i.e. vertices that are adjacent to at least $|K^D|/4$ vertices of K^D, and $I^- = I^D \setminus I^+$. We bound their sizes independently.

First, by counting the number n_e of edges between K^D and I^+ from the point of view of K^D, we get $n_e \leq |K^D| \cdot (\sqrt{2k}+1)$. From the point of view of I^+, we get $n_e \geq |K^D| \cdot |I^+|/4$. By combining the two inequalities, we get $|I^+| \leq 4(\sqrt{2k}+1)$.

It remains to show that $|I^-| \leqslant 6k + \sqrt{2k}$. To this end, we consider two types of vertices in K^D: those that are adjacent to more than $\sqrt{2k}$ edges of F in the solution, and the others. We then bound the number of vertices in I^- adjacent to (at least) a vertex of each type.

Since we add at most k edges to G, there are at most $\sqrt{2k}$ vertices in K^D incident to more than $\sqrt{2k}$ edges of F. By Observation 4.11, these vertices of K^D have at most $\sqrt{2k}(\sqrt{2k}+1) \leq 2k + \sqrt{2k}$ neighbors in I^D (and therefore in I^-).

To conclude the proof, it is thus sufficient to show that there are at most $4k$ vertices in I^- that are adjacent to vertices of K^D of the second type.

Let v be a vertex of K^D of the second type. We write $K_v = N(v) \cap K^D$ and $I_v^- = N(v) \cap I^-$. Observe that, by definition, $|K_v| \geqslant |K^D| - \sqrt{2k} \geq |K^D|/2$. Let $\bar{d}$ be the average degree in K_v of vertices in I_v^-. Since Rule 4.2-b cannot be applied, there are at least $|K_v| \cdot |I_v^-| - k$ edges between K_v and I_v^-, hence $\bar{d} \geqslant |K_v| - k/|I_v^-| \geqslant |K^D|/2 - k/|I_v^-|$. However, by definition of I^-, each vertex has degree at most $|K^D|/4$ in K_v hence $\bar{d} \leqslant |K^D|/4$. Combining the above yields $|I_v^-| \leqslant 4k/|K^D|$. Since there are at most $|K^D|$ vertices of the second type, the union of their neighborhoods has size at most $|K^D| \cdot 4k/|K^D| = 4k$, which is the sought result. ◀

5 Quadratic kernel for addition towards trivially perfect graphs

The goal of this section is to prove Theorem 1.3. Recall that trivially perfect graphs are (C_4, P_4)-free graphs. In what follows, we refer to induced P_4 or C_4 of a graph as its *obstructions*. We say that a pair (u, v) of vertices is a *diagonal* if $uv \notin E$ and there exists two vertices a, b such that $uavb$ is a P_4 or a C_4. Given a diagonal (u, v), the number of

obstructions containing (u, v) is the number of distinct pairs a, b such that $uavb$ is a P_4 or a C_4. Note that every obstruction contains exactly two diagonals and any solution must contain at least one of the two diagonals of each obstruction.

We first present a reduction rule that should be applied exhaustively, and then two reduction rules that should be applied once.

▶ **Rule 5.1.** *Let u, v be two non-adjacent vertices. If the number of obstructions containing u, v is at least $k + 1$, then add uv to E and decrease k by 1.*

▶ **Lemma 5.2.** *Rule 5.1 is safe.*

Moreover, Rule 5.1 can easily be applied in polynomial time.

The *modulator* $X(G)$ of G is the subset of vertices of G that are in at least one obstruction. Guo [21, Theorem 4] stated that (G, k) is a positive instance if and only if $(G[X(G)], k)$ is. Therefore, the following reduction rule is safe:

▶ **Rule 5.3.** *If $X(G) \neq V(G)$, remove all vertices of G that are not in $X(G)$.*

When the first two rules cannot be applied, we perform the following rule which detects trivially negative instances.

▶ **Rule 5.4.** *If $|V(G)| > 2k^2 + 2k$, output a trivially negative instance.*

To complete our proof, we simply have to prove that after applying the first two rules exhaustively, the size of a positive instance is quadratic. The next lemma ensures that Rule 5.4 is safe, which concludes the proof of Theorem 1.3.

▶ **Lemma 5.5.** *If (G, k) is a positive instance and every diagonal belongs to at most k obstructions, then $|X(G)| \leq 2k^2 + 2k$.*

6 Linear kernel for Starforest deletion

The goal of this section is to prove Theorem 1.4. Stars can be divided into two sets: centers and leaves. Let us define the notion of *center set* of a star forest.

▶ **Definition 6.1** (Center set). *Let $\mathbb{S}$ be a star-forest. A set $C^* \subseteq V(\mathbb{S})$ is a* center set *of $\mathbb{S}$ if C^* is a dominating set of $\mathbb{S}$ such that every star S of $\mathbb{S}$ contains exactly one vertex c of C^*. This vertex is called the center of S.*

Note that a center set is not necessarily unique since, in 2-stars, both vertices can be selected as a center. Given a star forest $\mathbb{S}$ with a set of centers C^*, the *leaves* of $\mathbb{S}$ are the vertices outside of C^*. By definition, every leaf has degree 1 and its unique neighbor is in C^*.

In what follows, we show how to use the structure of the input graph to identify and label vertices that are centers of an optimal solution, which leads to a Label-And-Reduce kernelization algorithm.

Let (G, k) be an instance of Starforest Deletion. Our first reduction rule, which is indeed safe, removes trivial connected components.

▶ **Rule 6.2** (Clean-up rule). *Remove from G any connected component with 1 or 2 vertices.*

Assume now that Rule 6.2 cannot be applied anymore.

▶ **Rule 6.3** (Center labeling rule). *Let C be the set of vertices of G that are adjacent to a vertex of degree 1 in G.*

(a) *For every $v \notin C$, if v is adjacent to a vertex u of C, delete all the other edges between v and C, and decrease the parameter accordingly.*

(b) *For every $u, v \in C$, if u and v are adjacent then remove uv from G and decrease k by 1.*

The fact that Rule 6.3 is safe is a consequence of the following lemma:

▶ **Lemma 6.4.** *Let C be the set of vertices of G that are adjacent to a vertex of degree 1. If (G, k) is a positive instance, then there exists a solution F of (G, k) and a center-set C^* of $G - F$ such that $C \subseteq C^*$.*

Lemma 6.4 ensures that Rule 6.3 is safe. Indeed, if there exists a solution, then there is also a solution where C is in the center-set. Hence we can safely remove all the edges between the vertices of C since each star only contains one vertex of the center-set of $G - F$. Moreover, if an edge between v and a vertex w of C is kept in $G - F$, then we can choose to keep any other edge between v and C instead of vw, since all the vertices of C are centers of their stars.

When neither Rule 6.2 nor Rule 6.3 can be applied, we apply the following rule:

▶ **Rule 6.5** (Center reduction rule). *Merge all the vertices of C, and remove all but $k + 2$ vertices of degree 1.*

▶ **Lemma 6.6.** *Rule 6.5 is safe.*

When Rules 6.2 to 6.5 cannot be applied, we apply the following rule once.

▶ **Rule 6.7** (Kernel size rule). *If $|V(G)| > 4k + 3$, return a trivial negative instance (e.g. $(P_4, 0)$). Otherwise, return (G, k).*

Rule 6.7 ensures that the returned kernel has at most $4k + 3$ vertices. In the remainder of this section, we study the structure of positive instances of Starforest Deletion to prove that Rule 6.7 is safe.

In the two following lemmas, we assume that none of Rules 6.2 to 6.5 can be applied. The following lemma uses the sparsity of starforests (they have many vertices of degree 1) to get information on the structure of positive instances.

▶ **Lemma 6.8.** *If (G, k) is a positive instance of Starforest Deletion with m edges, then G contains at least $m - 3k$ vertices of degree 1.*

In the last step of Rule 6.3, we remove all but $k + 2$ vertices of degree 1. In the following lemma, we apply Lemma 6.8 to show that the number of remaining vertices must be small.

▶ **Lemma 6.9.** *If (G, k) is a positive instance where no rule can be applied, then $|V(G)| \leq 4k + 3$.*

By applying the contrapositive of Lemma 6.9, we get that Rule 6.7 is safe.

Improving the multiplicative constant in the linear bound. In the proof of Lemma 6.9, we use a simple argument based on the minimum degree to show that the $3k$ remaining edges span at most $3k + 1$ vertices. The worst case is when every vertex has degree 2, that is, when every connected component is a cycle. In a cycle, an optimal solution can easily be

found in polynomial time, and therefore we can remove cycles. We can also show that long induced paths can be reduced. Combining these results gives a smaller kernel, at the cost of an increased running time and a slightly more involved analysis.

However, these improvements do not yield a sublinear kernel. It turns out that, under the Exponential Time Hypothesis, Starforest Deletion does not have a sublinear kernel. Indeed, Drange et al. [15] proved that, under ETH, Starforest Deletion does not admit a subexponential FPT algorithm, i.e. an algorithm running in time $O^*(2^{o(k)})$. Moreover, there is an $O^*(2^n)$ algorithm for Starforest Deletion: for each subset S of vertices, test whether there exists a solution in which S is the center set. Therefore, a kernel with $o(k)$ vertices would imply an $O^*(2^{o(k)})$ algorithm; a contradiction.

References

1 Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine learning*, 56(1):89–113, 2004.

2 Amir Ben-Dor, Ron Shamir, and Zohar Yakhini. Clustering gene expression patterns. *Journal of computational biology*, 6(3-4):281–297, 1999.

3 Ivan Bliznets, Marek Cygan, Pawel Komosa, Lukáš Mach, and Michał Pilipczuk. Lower bounds for the parameterized complexity of minimum fill-in and other completion problems. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1132–1151. SIAM, 2016.

4 Sebastian Böcker. A golden ratio parameterized algorithm for cluster editing. *Journal of Discrete Algorithms*, 16:79–89, 2012.

5 Pablo Burzyn, Flavia Bonomo, and Guillermo Durán. Np-completeness results for edge modification problems. *Discrete Applied Mathematics*, 154(13):1824–1844, 2006.

6 Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996.

7 Yixin Cao and Jianer Chen. Cluster editing: Kernelization based on edge cuts. *Algorithmica*, 64(1):152–169, 2012.

8 Yixin Cao and Yuping Ke. Improved kernels for edge modification problems. *arXiv preprint arXiv:2104.14510*, 2021.

9 Jianer Chen and Jie Meng. A 2k kernel for the cluster editing problem. *Journal of Computer and System Sciences*, 78(1):211–220, 2012.

10 Christophe Crespelle, Pål Grønås Drange, Fedor V Fomin, and Petr A Golovach. A survey of parameterized algorithms and the complexity of edge modification. *arXiv preprint arXiv:2001.06867*, 2020.

11 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5. Springer, 2015.

12 Peter Damaschke and Olof Mogren. Editing the simplest graphs. In *International Workshop on Algorithms and Computation*, pages 249–260. Springer, 2014.

13 Pål Grønås Drange, Fedor V Fomin, Michał Pilipczuk, and Yngve Villanger. Exploring the subexponential complexity of completion problems. *ACM Transactions on Computation Theory (TOCT)*, 7(4):1–38, 2015.

14 Pål Grønås Drange and Michał Pilipczuk. A polynomial kernel for trivially perfect editing. *Algorithmica*, 80(12):3481–3524, 2018.

15 Pål Grønås Drange, Felix Reidl, Fernando Sánchez Villaamil, and Somnath Sikdar. Fast biclustering by dual parameterization. *arXiv preprint arXiv:1507.08158*, 2015.

16 Mael Dumas, Anthony Perez, and Ioan Todinca. A cubic kernel for trivially perfect edition. *private communication.*

17 Fedor V Fomin, Stefan Kratsch, Marcin Pilipczuk, Michał Pilipczuk, and Yngve Villanger. Tight bounds for parameterized complexity of cluster editing with a small number of clusters. *Journal of Computer and System Sciences*, 80(7):1430–1447, 2014.

18 Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: theory of parameterized preprocessing*. Cambridge University Press, 2019.

19 Esha Ghosh, Sudeshna Kolay, Mrinal Kumar, Pranabendu Misra, Fahad Panolan, Ashutosh Rai, and MS Ramanujan. Faster parameterized algorithms for deletion to split graphs. *Algorithmica*, 71(4):989–1006, 2015.

20 Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Graph-modeled data clustering: Exact algorithms for clique generation. *Theory of Computing Systems*, 38(4):373–392, 2005.

21 Jiong Guo. Problem kernels for np-complete edge deletion problems: Split and related graphs. In *International Symposium on Algorithms and Computation*, pages 915–926. Springer, 2007.

22 Peter L Hammer and Bruno Simeone. The splittance of a graph. *Combinatorica*, 1(3):275–284, 1981.

23 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

24 Yunlong Liu, Jianxin Wang, Jie You, Jianer Chen, and Yixin Cao. Edge deletion problems: Branching facilitated by modular decomposition. *Theoretical Computer Science*, 573:63–70, 2015.

25 Federico Mancini. Graph modification problems related to graph classes. *PhD degree dissertation, University of Bergen Norway*, 2, 2008.

26 Assaf Natanzon, Ron Shamir, and Roded Sharan. Complexity classification of some edge modification problems. *Discrete Applied Mathematics*, 113(1):109–128, 2001.

27 René van Bevern, Vincent Froese, and Christian Komusiewicz. Parameterizing edge modification problems above lower bounds. *Theory of Computing Systems*, 62(3):739–770, 2018.

28 Zhenyu Wu and Richard Leahy. An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 15(11):1101–1113, 1993.

29 Mihalis Yannakakis. Node-and edge-deletion np-complete problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 253–264, 1978.

Parameterized Complexities of Dominating and Independent Set Reconfiguration

Hans L. Bodlaender
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Carla Groenland
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Céline M. F. Swennenhuis
Department of Mathematics and Computer Science,
Eindhoven University of Technology, The Netherlands

Abstract

We settle the parameterized complexities of several variants of independent set reconfiguration and dominating set reconfiguration, parameterized by the number of tokens. We show that both problems are XL-complete when there is no limit on the number of moves and XNL-complete when a maximum length ℓ for the sequence is given in binary in the input. The problems are known to be XNLP-complete when ℓ is given in unary instead, and W[1]- and W[2]-hard respectively when ℓ is also a parameter. We complete the picture by showing membership in those classes.

Moreover, we show that for all the variants that we consider, token sliding and token jumping are equivalent under pl-reductions. We introduce partitioned variants of token jumping and token sliding, and give pl-reductions between the four variants that have precise control over the number of tokens and the length of the reconfiguration sequence.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Parameterized complexity, independent set reconfiguration, dominating set reconfiguration, W-hierarchy, XL, XNL, XNLP

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.9

Related Version *Full Version*: `https://arxiv.org/abs/2106.15907` [2]

Funding *Carla Groenland*: Supported by the project CRACKNP that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research innovation programme (grant agreement No 853234).
Céline M. F. Swennenhuis: Supported by the Netherlands Organization for Scientific Research under project no. 613.009.031b.

Acknowledgements We would like to thank the referees for useful comments.

1 Introduction

In this paper, we study the parameterized complexity of reconfiguration of independent sets, and of dominating sets, with the sizes of the sets as parameter. Interestingly, the complexity varies depending on the assumptions on the length of the reconfiguration sequence, which can be unbounded, given in binary, given in unary, or given as second parameter. One can study the reconfiguration problems for different reconfiguration rules; we will show equivalence regarding the complexity for several reconfiguration rules.

Independent Set Reconfiguration

In the Independent Set Reconfiguration problem, we are given a graph and two independent sets A and B, and wish to decide we can "reconfigure" A to B via a "valid" sequence of independent sets $A, I_1, \ldots, I_{\ell-1}, B$. Suppose that we represent the current independent set by placing a token on each vertex. We can move between two independent sets by moving a single token. We consider two well-studied rules for deciding how we can move the tokens.

- Token jumping (TJ): we can "jump" a single token to any vertex that does not yet contain a token.
- Token sliding (TS): we can "slide" a single token to an adjacent vertex that does not yet contain a token.

Independent Set Reconfiguration is PSPACE-complete for both rules [9], but their complexities may be different when restricting to specific graph classes. For example, Independent Set Reconfiguration is NP-complete on bipartite graphs under the token jumping rule, but remains PSPACE-complete under the token sliding rule [11].

There is a third rule which has been widely studied, called the token addition-removal rule, but this rule is equivalent to the token jumping rule for our purposes (see e.g. [9, Theorem 1]). As further explained later, we show that the token jumping and the token sliding rule are also equivalent in some sense (which is much weaker but does allow us to control all the parameters that we care about). We will therefore not explicitly mention the specific rule under consideration below.

Throughout this paper, our reconfiguration problems are parameterized by the number of tokens (the size of the independent set). Independent Set Reconfiguration is W[1]-hard [7], but the problem is not known to be in W[1]. We show that in fact it is complete for the class XL, consisting of the parameterized problems that can be solved by a deterministic algorithm that uses $f(k) \log n$ space, where k is the parameter, n the input size and f any computable function.

▶ **Theorem 1.** *Independent Set Reconfiguration is* XL*-complete.*

In the Timed Independent Set Reconfiguration, we are given an integer ℓ in unary and two independent sets A and B in a graph G, and need to decide whether there is a reconfiguration sequence from A to B of length at most ℓ. We again parameterize it by the number of tokens. The following result has been shown by the authors and Nederlof [1].

▶ **Theorem 2** ([1])**.** *Timed Independent Set Reconfiguration is* XNLP*-complete.*

The class XNLP (also denoted $N[f \operatorname{poly}, f \log]$ by Elberfeld et al. [5]) is the class of parameterized problems that can be solved with a non-deterministic algorithm with simultaneously, the running time bounded by $f(k)n^c$ and the space usage bounded by $f(k) \log n$, with k the parameter, n the input size, c a constant, and f a computable function. This is a natural subclass of the class XNL, which consists of the parameterized problems that can be solved by a nondeterministic algorithm that uses $f(k) \log n$ space. Amongst others, XNL was studied by Chen et al. [3].

The classes XL, XNL, XSL, XP can be seen as the parameterized counterparts of L, NL, SL, P respectively. Although no explicit time bound is given, we can freely add a time bound of $2^{f(k) \log n}$, and thus XNL is a subset of XP. We remark that XL=XSL[1] [15] (just as L=SL), XL⊆XNL and XNLP⊆XNL.

[1] A proof can be found in Appendix A of the full version [2]

Table 1 The table shows the parameterized complexities of the independent set and dominating set reconfiguration problems, parameterized by the number of tokens, depending on the treatment of the bound ℓ on the length of the reconfiguration sequence.

Sequence length ℓ	Independent Set	Dominating Set	Sources
not given	XL-complete	XL-complete	Theorems 1, 5
parameter	W[1]-complete	W[2]-complete	Theorems 4, 5 and [12]
unary input	XNLP-complete	XNLP-complete	[1]
binary input	XNL-complete	XNL-complete	Theorems 3, 5

In Binary Timed Independent Set Reconfiguration, the bound ℓ on the length of the sequence is given in binary[2]. Interestingly, this slight adjustment to Timed Independent Set Reconfiguration is complete for XNL instead.

▶ **Theorem 3.** *Binary Timed Independent Set Reconfiguration is* XNL*-complete.*

Finally, we consider what happens when we consider ℓ to be a parameter instead. If Timed Independent Set Reconfiguration (or equivalently Binary Timed Independent Set Reconfiguration) is parameterized by the size of the independent set and the length of the sequence[3], then it is W[1]-hard [12][4]. We show that in this case, W[1] is the "correct class".

▶ **Theorem 4.** *Timed Independent Set Reconfiguration is in* W[1] *when parameterized by the size of the independent set and the length of the sequence.*

Dominating set reconfiguration

The dominating set reconfiguration problem is similar to the independent set reconfiguration problem, but in this case all sets in the sequence must form a dominating set in the graph. This again gives a PSPACE-complete problem [6]. We define the parameterized problems Dominating Set Reconfiguration, Timed Dominating Set Reconfiguration and Binary Timed Dominating Set Reconfiguration similarly as their independent set counterparts, again parameterized by the number of tokens. Since Dominating Set is W[2]-complete and Independent Set is W[1]-complete (parameterized by "the number of tokens"), it may be expected that the reconfiguration variants also do not have the same parameterized complexity. Indeed, Timed Dominating Set Reconfiguration is W[2]-hard when it is moreover parameterized by the length of the sequence [12].

We show that the picture is otherwise the same as for independent set.

▶ **Theorem 5.** *Dominating Set Reconfiguration is* XL*-complete. Binary Timed Dominating Set Reconfiguration is* XNL*-complete. Timed Dominating Set Reconfiguration is* W[2]*-complete when moreover parameterized by the length of the sequence.*

It was already known that Timed Dominating Set Reconfiguration is XNLP-complete [1]. The proof Theorem 5 can be found in Appendix C of the the full version of this paper [2]. A summary of our results can be found in Table 1.

[2] Giving ℓ in binary implies that it contributes $\log_2(\ell)$ to the size of an instance of Binary Timed Independent Set Reconfiguration.

[3] We can also consider it to be parameterized by the sum of the two parameters.

[4] Mouawad et al. [12] only studied the token jumping variant, but Theorem 6 implies the hardness also holds for token sliding.

Many other types of reconfiguration problems have been studied as well, and we refer the reader to the surveys by Van den Heuvel [14] and Nishimura [13] for further background.

Equivalences between token jumping and token sliding

In Appendix D of the full version [2], we introduce partitioned variants of token sliding and token jumping in which the tokens need to stay within specified token sets. We prove the theorem below by giving reductions from and to the independent set reconfiguration problems (with the four rules: (partitioned) token sliding and (partitioned) token jumping) that control the number of tokens and the length of the reconfiguration sequence. We give similar reductions for the dominating set reconfiguration problems.

▶ **Theorem 6.** *For the following parameterized problems, their variant with the token jumping rule is equivalent under pl-reductions and fpt-reductions to their variant with the token sliding rule:* Independent Set Reconfiguration, Timed Independent Set Reconfiguration, Binary Independent Set Reconfiguration *and* Timed Independent Set Reconfiguration *when moreover parameterized by the length of the sequence. The same holds for the dominating set variants.*

2 Preliminaries

We write $\mathbf{N}$ for the set of integers $0, 1, 2, \ldots$ and write $[a, b]$ for the set of integers x with $a \leq x \leq b$. All logs in this paper are base 2.

Parameterized reductions

A *parameterized reduction* from a parameterized problem $Q_1 \subseteq \Sigma_1^* \times \mathbf{N}$ to a parameterized problem $Q_2 \subseteq \Sigma_2^* \times \mathbf{N}$ is a function $f : \Sigma_1^* \times \mathbf{N} \to \Sigma_2^* \times \mathbf{N}$, such that the following holds.

1. For all $(x, k) \in \Sigma_1^* \times \mathbf{N}$, $(x, k) \in Q_1$ if and only if $f((x, k)) \in Q_2$.
2. There is a computable function g, such that for all $(x, k) \in \Sigma_1^* \times \mathbf{N}$, if $f((x, k)) = (y, k')$, then $k' \leq g(k)$.

A *parameterized logspace reduction* or *pl-reduction* is a parameterized reduction for which there is an algorithm that computes $f((x, k))$ in space $\mathcal{O}(g(k) + \log n)$, with g a computable function and $n = |x|$ the number of bits to denote x.

Symmetric Turing Machine

A Symmetric Turing Machine (STM) is a Nondeterministic Turing Machine (NTM), where the transitions are symmetric. That means that for any transition, we can also take its inverse back. More formally, a Symmetric Turing Machine with one work tape is a 5-tuple $(\mathcal{S}, \Sigma, \mathcal{T}, s_{\mathsf{start}}, \mathcal{A})$, where $\mathcal{S}$ is a finite set of *states*, Σ is the *alphabet*, $\mathcal{T}$ is the set of *transitions*, s_{start} is the *start state* and $\mathcal{A}$ is the set of *accepting states*. A transition $\tau \in \mathcal{T}$ is a tuple of the form (p, Δ, q) describing a transition the STM may take, where $p, q \in \mathcal{S}$ are states and Δ is a *tape triple*. A tape triple is equal to either (ab, δ, cd), where $a, b, c, d \in \Sigma$ and $\delta \in \{-1, 1\}$, or $(a, 0, b)$, where $a, b \in \Sigma$. For example, the transition $(p, (ab, 1, cd), q)$ describes that if the STM is in state p, reads a and b on the current work tape cell and the cell directly right of it, then it can replace a with c, b with d, moving the head to the right and going to state q.

Let $\Delta = (ab, \delta, cd)$ be a state triple, then its inverse is defined as $\Delta^{-1} = (cd, -\delta, ab)$. The inverse of $\Delta = (a, 0, b)$ is defined as $\Delta^{-1} = (b, 0, a)$. By definition of the Symmetric Turing Machine, for any $\tau \in \mathcal{T}$, there is an *inverse* transition $\tau^{-1} \in \mathcal{T}$, i.e. if $\tau = (p, \Delta, q) \in \mathcal{T}$, then $\tau^{-1} = (q, \Delta^{-1}, p) \in \mathcal{T}$.

We say that STM $\mathcal{M}$ *accepts* if there is a computation of $\mathcal{M}$ that ends in an accepting state. We remark that the Turing Machines in this paper do not have an input tape, as it is hidden in the states (see Appendix A of the full version [2]). For a more formal definition of Symmetric Turing Machines we would like to refer to the definition from Louis and Papadimitriou in [10].

Note that we may assume that there is only one accepting state $s_{\mathsf{acc}} \in \mathcal{A}$, by creating this new state s_{acc} and adding a transition to s_{acc} from any original accepting state. We may also assume all transitions to move the tape head to the left or right. This can be accomplished by replacing each transition $\tau = (p, (a, 0, b), q)$ with $2|\Sigma|$ transitions as follows. For all $\sigma \in \Sigma$, we create a new state s_σ and two new transitions $\tau_\sigma^1 = (p, (a\sigma, 1, b\sigma), s_\sigma)$ and $\tau_\sigma^2 = (s_\sigma, (b\sigma, -1, b\sigma), q)$.

The following problem will be used in the reductions of Section 3.

Accepting Log-Space Symmetric Turing Machine
Given: A STM $\mathcal{M} = (\mathcal{S}, \Sigma, \mathcal{T}, s_{\mathsf{start}}, \mathcal{A})$ with $\Sigma = [1, n]$ and a work tape with k cells.
Parameter: k.
Question: Does $\mathcal{M}$ accept?

We define Accepting Log-Space Nondeterministic Turing Machine to be the Nondeterministic Turing Machine analogue of Accepting Log-Space Symmetric Turing Machine.

▶ **Theorem 7.** Accepting Log-Space Symmetric Turing Machine *is* XL*-complete and* Accepting Log-Space Nondeterministic Turing Machine *is* XNL*-complete.*

A proof of Theorem 7 can be found in Appendix A of the full version [2].

In our reductions we use the notion of a *configuration*, describing exactly in what state an NTM (and therefore an STM) and its tape are.

▶ **Definition 8.** *Let $\mathcal{M} = (\mathcal{S}, \Sigma, \mathcal{T}, s_{\mathsf{start}}, \mathcal{A})$ be an NTM with $\Sigma = [1, n]$ and k cells on the work tape and let $\alpha \in \Sigma^*$ be the input. A* configuration *of $\mathcal{M}$ is a $k+2$ tuple $(p, i, \sigma_1, \ldots, \sigma_k)$ where $p \in \mathcal{S}$, $i \in [1, k]$ and $\sigma_1, \ldots, \sigma_k \in \Sigma$, describing the state, head position and content of the work tape of $\mathcal{M}$ respectively.*

3 Proof of Theorem 1: XL-completeness

By Theorem 6, it suffices to show that the following problem is XL-complete.

Partitioned TS-Independent Set Reconfiguration
Given: Graph $G = (V, E)$; independent sets $I_{\mathsf{init}}, I_{\mathsf{fin}}$ of size k; a partition $V = \sqcup_{i=1}^{k} P_i$ of the vertex set.
Parameter: k.
Question: Does there exist a sequence $I_{\mathsf{init}} = I_0, I_1, \ldots, I_T = I_{\mathsf{fin}}$ of independent sets of size k for some T, with $|I_t \cap P_i| = 1$ for all $t \in [0, T]$ and $i \in [1, k]$, such that for all $t \in [1, T]$, $I_t = I_{t-1} \setminus \{u\} \cup \{v\}$ for some $uv \in E(G)$ with $u \in I_{t-1}$ and $v \notin I_{t-1}$?

Theorem 6 then implies the XL-completeness results for the other variants of Independent Set Reconfiguration.

The XL-completeness proof for Partitioned TS-Dominating Set Reconfiguration is similar and given in Appendix C of the full version [2].

▶ **Theorem 9.** Partitioned TS-Independent Set Reconfiguration *is* XL*-complete.*

Proof. By Theorem 7, it suffices to give pl-reductions to and from Accepting Log-Space Symmetric Turing Machine.

The problem is in XL (=XSL) as it can be simulated with a Symmetric Turing Machine with $\mathcal{O}(k \log n)$ space as follows. We store the current independent set of size k on the work tape, which takes about $k \cdot \log n$ bits space. We use the transitions of the STM to model the changes of one of the vertices in the independent set. For all vertices $u, v \in V$, we have a sequence of states and transitions that allows you to remove u and add v to the independent set currently stored on the work tape, if the following assumptions are met: $u \in I$, $v \notin I$, $uv \in E(G)$, u, v are part of the same token set (part of the partition) and I' is an independent set. This gives a total of $O(n^2k^2)$ states. There is one accepting state, reachable via a sequence of states and transitions that verifies that the current independent set is the final independent set. All transitions are symmetric.

We prove the problem to be XL-hard by giving a reduction from Accepting Log-Space Symmetric Turing Machine. Let $\mathcal{M} = (\mathcal{S}, \Sigma, \mathcal{T}, s_{\mathsf{start}}, \mathcal{A})$ be the STM of a given instance, with $\mathcal{A} = \{s_{\mathsf{acc}}\}$, $\Sigma = [1, n]$ and a work tape of k cells. We may assume that $\mathcal{M}$ only accepts if the symbol 1 is on every cell of the work tape and the head is at the first position. This can be done by creating a new accepting state and adding $\mathcal{O}(k)$ transitions from s_{acc} to this new state, which set only 1's on the work tape and move the head to the first position. We create an instance Γ of Partitioned TS-Independent Set Reconfiguration with $k' = k + 1$ tokens. These tokens will simulate the configuration of $\mathcal{M}$: k tape-tokens modeling the work tape cells and one state token describing the current state and tape head position.

Tape gadgets. For each work tape cell $i \in [1, k]$, we create a *tape gadget* consisting of $n + 1$ vertices as follows. We add a vertex v^i_σ for all $\sigma \in \Sigma = [1, n]$ and a vertex y^i, connected to v^i_σ for all $\sigma \in \Sigma$. The vertices in a tape gadget form a token set (a part of the partition, i.e. exactly one of these $n + 1$ vertices is in the independent set at any given time). The symbol σ that is on the ith work tape cell of $\mathcal{M}$ is simulated by which v^i_σ is in the independent set.

State vertices. The last token set is the set of all transition and state vertices (defined below), meaning that exactly one of these vertices is in the independent set at any given time. The token of this set (called the "state token") simulates the state of $\mathcal{M}$, the position of the head, and the transition $\mathcal{M}$ takes.

We create a vertex p^i for each state $p \in \mathcal{S}$ and all head positions $i \in [1, k]$. We add edges $p^i y^{i'}$ for all $i' \in [1, k]$. These vertices will simulate the current state of $\mathcal{M}$ and the position i of the tape head.

Transition vertices. To go from one state vertex to another, we create a path of three transition vertices, to checking whether the work tape agrees with the transition before and afterwards, and one allowing moving some tokens of the tape gadget. To control when we can move tokens in the tape gadgets, we put edges between y^i and all state and transition vertices (for each $i \in [1, k]$), unless specified otherwise. We further outline which edges are present below, and give an example in Figure 1.

Recall that we may assume that head always moves left or right. Consider first a transition $\tau \in \mathcal{T}$ that moves the head to the right, say $\tau = (p, (ab, 1, cd), q)$. For all $i \in [1, k-1]$, we create a path between state vertices p^i and q^{i+1} consisting of three "transition" vertices: τ^i_{ab}, τ^i_{shift} and τ^i_{cd}. In order to ensure a token can only be on τ^i_{ab} when the token on the ith tape gadget represents the symbol a, we add edges from τ^i_{ab} to v^i_σ for all $\sigma \in \Sigma \setminus \{a\}$. Similarly,

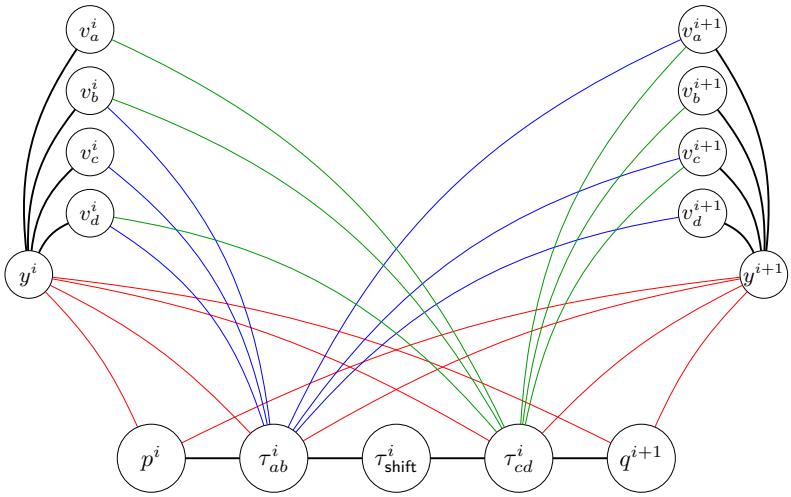


Figure 1 Sketch of part of the construction of Theorem 9. Given are the two tape gadgets for positions i and $i+1$, two state vertices and a transition path for transition $\tau = (p, (ab, 1, cd), q)$.

edges to v_σ^{i+1} are added for all $\sigma \in \Sigma \setminus \{b\}$, as well as edges between τ_{cd}^i and all v_σ^i and v_σ^{i+1} except for v_c^i and v_d^{i+1}. When the token is on the shift vertex τ_{shift}^i, the independent set is allowed to change the token in the ith and $(i+1)$th tape gadget. Therefore, we remove the edges between y^i and y^{i+1} and τ_{shift}^i.

Note that the constructed paths also handle transitions which move the tape head to the left, as we can transverse the constructed paths in both directions. We omit the details.

Initial and final independent sets. Recall that s_{start} is the starting state of $\mathcal{M}$. We let the initial independent set be $I_{\mathsf{init}} = \{s_{\mathsf{start}}^1\} \cup \left(\bigcup_{i\in[1,k]}\{v_1^i\}\right)$, corresponding to the initial configuration of $\mathcal{M}$. Let the final independent set be $I_{\mathsf{fin}} = \{s_{\mathsf{acc}}^1\} \cup \left(\bigcup_{i\in[1,k]}\{v_1^i\}\right)$, where s_{acc} is the accepting state of $\mathcal{M}$. We note that both I_{init} and I_{fin} are independent sets.

Let Γ be the created instance of Partitioned TS-Independent Set Reconfiguration. We prove that Γ is a yes-instance is and only if $\mathcal{M}$ accepts. We use the following function, hinting at the equivalence between configurations of $\mathcal{M}$ and certain independent sets of Γ.

▶ **Definition 10.** *Let $C = (p, i, \sigma_1, \ldots, \sigma_k)$ be a configuration of $\mathcal{M}$. Then let $I(C)$ be the corresponding independent set in Γ:*

$$I(C) = \{p^i\} \cup \left(\bigcup_{j=1}^{k} \left\{v_{\sigma_j}^j\right\}\right)$$

▷ Claim 11. Γ is a yes-instance if $\mathcal{M}$ accepts.

Proof. Let $C_1, \ldots, C_\ell$ be the sequence of configurations such that $\mathcal{M}$ accepts. We note that $I(C_1) = I_{\mathsf{init}}$. For each $t \in [1, \ell - 1]$, we do the following. Let $C_t = (p, i, \sigma_1, \ldots, \sigma_k)$. Let τ be the transition $\mathcal{M}$ takes to the next configuration. Assume $\tau = (p, (ab, 1, cd), q)$, the case where τ moves the head to the left will be discussed later, but is similar. Take the following

sequence of independent sets, where $I(C_t) = I_0$ is the current independent set:

$$I_1 = I_0 \setminus \{p^i\} \cup \{\tau^i_{ab}\} \qquad I_5 = I_4 \setminus \{v^{i+1}_b\} \cup \{y^{i+1}\}$$
$$I_2 = I_1 \setminus \{\tau^i_{ab}\} \cup \{\tau^i_{\mathsf{shift}}\} \qquad I_6 = I_5 \setminus \{y^{i+1}\} \cup \{v^{i+1}_d\}$$
$$I_3 = I_2 \setminus \{v^i_a\} \cup \{y^i\} \qquad I_7 = I_6 \setminus \{\tau^i_{\mathsf{shift}}\} \cup \{\tau^i_{cd}\}$$
$$I_4 = I_3 \setminus \{y^i\} \cup \{v^i_c\} \qquad I_8 = I_7 \setminus \{\tau^i_{cd}\} \cup \{q^{i+1}\}$$

Notice that this sequence of independent sets is allowed, as all sets are independent, each token stays within its token set and each next independent set is a slide away from its previous. Also, we see that $I(C_{t+1}) = I_8$. For transition $\tau = (p, (ab, -1, cd), q)$, where the tape head moves to the left, we do the following. Let $\tau^{-1} = (q, (cd, 1, ab), p)$ be the inverse. We take the sequence that belongs to τ^{-1} backwards, i.e. if $I_0, \ldots, I_8$ was the sequence of independent sets as described for τ^{-1}, then take the sequence $I_8, \ldots, I_0$.

We note that $I(C_\ell) = I_{\mathsf{fin}}$ is the final independent set, as we assumed the machine only to accept with $\sigma_i = 1$ for all $i \in [1, k]$ and the head at the first position. Therefore, we find that this created sequence of independent sets is a solution to Γ. ◁

We now prove the other direction.

▷ Claim 12. $\mathcal{M}$ accepts if Γ is a yes-instance.

Proof. Let $I_{\mathsf{init}} = I_1, \ldots, I_{\ell-1}, I_\ell = I_{\mathsf{fin}}$ be the sequence of independent sets that is a solution to Γ. We assume this sequence to be minimal, implying that no independent set can occur twice.

The state token should always be on either a state or transition vertex, because of its token set. Let $I'_1, \ldots, I'_{\ell'}$ be the subsequence of $I_1, \ldots, I_\ell$ of independent sets that include a state vertex. We will prove that the configurations of $\mathcal{M}$, simulated by this subsequence, is a sequence of configurations that leads to the accepting state s_{acc}. To do this, first we note some general facts about I_t for $t \in [1, \ell]$.

If the state token of I_t is on a state vertex p^i, then I_{t+1} slides the state token to a neighbor of p^i. This is because all $y^{i'}$ for $i' \in [1, k]$ are neighbors of p^i, hence the tokens in the tape gadgets are on some $v^{i'}_\sigma$ and cannot move. The same holds for transition vertices of the form τ^i_{ab}. If $\tau^i_{\mathsf{shift}} \in I_t$, then y^i and y^{i+1} are not neighbors of the state token. Therefore, the ith and $i+1$th tape gadgets token can now slide. If $\tau^i_{ab} \in I_t$, then $v^i_a \in I_t$ and $v^{i+1}_b \in I_t$. This is because all other vertices of the ith and $i+1$th tape gadgets are neighbors of τ^i_{ab}.

Recall that $I'_1, \ldots, I'_{\ell'}$ is the sequence of independent sets with the state token on a state vertex. For any I'_t with $t \in [1, \ell']$, let C_t be the unique configuration of $\mathcal{M}$ such that $I(C_t) = I'_t$. We prove that $C_1, \ldots, C_{\ell'}$ is an allowed sequence of configurations for $\mathcal{M}$. Note that this implies that $\mathcal{M}$ accepts as $C_{\ell'}$ is the accepting configuration.

We fix $t \in [1, \ell']$ and focus on the transition between C_t and C_{t+1}. Let $A_1, \ldots, A_R$ be the sequence of independent sets in the solution of Γ, that are visited between I'_t and I'_{t+1}. By definition of I'_t and I'_{t+1}, A_r does not contain a state vertex for all $r \in [1, R]$, therefore each A_r must have its state token on a transition vertex. Each such transition vertex corresponds to the same transition $\tau = (p, \Delta, q)$, as this is the only path the state token can take. We assume that $\Delta = (ab, 1, cd)$, the case $\Delta = (ab, -1, cd)$ can be proved with similar arguments. The set A_1 contains transition vertex τ^i_{ab} and therefore I'_t contains v^i_a and v^{i+1}_b. Also, A_R contains τ^i_{cd}, implying that $v^i_c, v^{i+1}_d \in I'_{t+1}$. We note that $A_2, \ldots, A_{R-1}$ must contain τ^i_{shift}: only the ith and $i+1$th tape gadget tokens can shift when the state token is on τ^i_{shift}. So if the state token would be on τ^i_{ab} or τ^i_{cd} twice in $A_1, \ldots, A_R$, the independent sets would be equal, contradicting the minimal length of the sequence.

Combining this all, we conclude that if $I(C_t) = I'_t$ and $I(C_{t+1}) = I'_{t+1}$, there is an allowed sequence of independent set, traversing the path belonging to a transition $\tau = (p, (ab, 1, cd), q)$. Therefore, $I_{t+1} = I_t \setminus \{v_a^i, v_b^{i+1}, p^i\} \cup \{v_c^i, v_d^{i+1}, q^{i+1}\}$ and we are allowed to take transition τ from C_t to end up in configuration C_{t+1}. ◁

Hence, Γ is a yes instance if and only if $\mathcal{M}$ accepts and we find that the given reduction is correct. This concludes the proof of Theorem 9. ◀

4 Proof of Theorem 3: XNL-completeness

In this section we prove Theorem 3 by showing that the following problem is XNL-complete.

Binary Timed Partitioned TS-Independent Set Reconfiguration
Given: Graph $G = (V, E)$; independent sets $I_{\mathsf{init}}, I_{\mathsf{fin}}$ of size k; integer ℓ given in binary; a partition $V = \sqcup_{i=1}^{k} P_i$ of the vertex set.
Parameter: k.
Question: Does there exist a sequence $I_{\mathsf{init}} = I_0, I_1, \ldots, I_T = I_{\mathsf{fin}}$ of independent sets of size k with $T \leq \ell$ and $|I_t \cap P_i| = 1$ for all $t \in [0, T]$ and $i \in [1, k]$, such that for all $t \in [1, T]$, $I_t = I_{t-1} \setminus \{u\} \cup \{v\}$ for some $uv \in E(G)$ with $u \in I_{t-1}$ and $v \notin I_{t-1}$?

To prove XNL-hardness, we introduce a variant of CNF-SAT. The following is a "long chain"-variant of the XNLP-complete problems "chained CNF-Satisfiability" introduced by [1].

Long Partitioned Positive Chain Satisfiability
Input: Integers $k, q, r \in \mathbf{N}$ with r given in binary and $r \leq q^k$; Boolean formula F, which is an expression on $2q$ positive variables and in conjunctive normal form; a partition of $[1, q]$ into k parts $P^1, \ldots, P^k$.
Parameter: k.
Question: Do there exist variables $x_j^{(t)}$ for $t \in [1, r]$ and $j \in [1, q]$, such that we can satisfy the formula

$$\bigwedge_{1 \leq t \leq r-1} F(x_1^{(t)}, \ldots, x_q^{(t)}, x_1^{(t+1)}, \ldots, x_q^{(t+1)})$$

by setting, for $i \in [1, k]$ and $t \in [1, r]$, exactly one variable from the set $\{x_j^{(t)} : j \in P_i\}$ to true and all others to false?

We remark that all XNLP-complete "chained satisfiability" variant of [1] have an XNL-complete analogue, but we decided to only present the form we need for this section. In Appendix B of the full version [2], we prove the following result.

▶ **Theorem 13.** Long Partitioned Positive Chain Satisfiability *is* XNL-*complete.*

From this, we derive the following result.

▶ **Theorem 14.** Binary Timed Partitioned TS-Independent Set Reconfiguration *is* XNL-*complete.*

Recall that Theorem 6 then implies the XNL-completeness results for the other variants of Binary Timed Independent Set Reconfiguration. A similar proof for the dominating set variant can be found in Appendix C.2 of the full version [2].

Proof of Theorem 14. We first show that Binary Timed Partitioned TS-Independent Set Reconfiguration is in XNL, that is, it can be modelled by a Nondeterministic Turing Machine using a work tape of size $\mathcal{O}(k \log n)$. One can store the current independent set of size k on the work tape and allow only transitions between an independent set I to an independent set $I' = I \setminus \{v\} \cup \{w\}$ if $vw \in E$, $v \in I$ and $w \notin I$. We can generate the possible independent sets adjacent to a given independent set I and keep track of the number of moves on a work tape of size $\mathcal{O}(k \log n)$. Since the number of independent sets of size k is at most n^k, and a shortest sequence consists of distinct independent sets, we may assume that $\ell \leq n^k$.

To prove that Binary Timed Partitioned TS-Independent Set Reconfiguration is XNL-hard, we give a reduction from Long Partitioned Positive Chain Satisfiability. The construction is similar (but more cumbersome) way as the one in [1, Theorem 4.11].

Let $(q, r, F, P^1, \ldots, P^k)$ be an instance of Long Partitioned Positive Chain Satisfiability. We will create an instance Γ of Binary Timed Partitioned TS-Independent Set Reconfiguration with $3k + 1$ token sets. The idea is to represent the choice of which variables $x_j^{(t)}$ are set to true with variable gadgets, and to create a clause checking gadget that verifies that $F(x_1^{(t)}, \ldots, x_q^{(t)}, x_1^{(t+1)}, \ldots, x_q^{(t+1)})$ is true. The time counter gadget has k tokens, which together represent the integer t. Using the time constraint, we ensure that we have to follow a very specific sequence of moves, and can therefore not change which $x_j^{(t)}$ is true after we passed an independent set that made a choice for this.

Time counter gadget. We create k *time tokens* who have its token set within the time counter gadget, where the positions of these tokens represent an integer $t \in [1, r]$ with $r \leq q^k$. We create k timers, consisting each of $4q$ vertices. For $i \in [1, k]$, the timer t^i is a cycle on vertices $t_0^i, \ldots, t_{4q-1}^i$, which forms a token set for one of the time tokens. If the time tokens are on the vertices $t_{\ell_1}^1, \ldots, t_{\ell_k}^k$, then this represents the current time as

$$t = \sum_{i=1}^{k} (\ell_i \bmod q) q^{i-1}.$$

Henceforth, we will silently assume t to be given by the position of the time tokens as specified above. How these timers are connected such that they work as expected will be discussed later.

Variable gadget. We create four sets $A = \{a_1, \ldots, a_q\}$, $B = \{b_1, \ldots, b_q\}$, $C = \{c_1, \ldots, c_q\}$ and $D = \{d_1, \ldots, d_q\}$ that all contain q vertices. These sets will be used to model which variables $x_j^{(t)}$ are chosen to be true. We partition the sets in the same way as the variables, setting $A^i = \{a_j : j \in P^i\}$ for all $i \in [1, k]$ and defining B^i, C^i and D^i similarly.

For all $i \in [1, k]$, we make (A^i, B^i) and (C^i, D^i) complete bipartite graphs, adding the edges $a_j b_{j'}$ and $c_j d_{j'}$ for all $j, j' \in P^i$. We specify $A^i \cup B^i$ and $C^i \cup D^i$ as token sets, and refer to the corresponding $2k$ tokens as *variable tokens*. The first set is used to model the choice of the true variable $x_j^{(t)}$ for $j \in P^i$ for all odd t, whereas the second partition models the same for any even t.

We will enforce the following. Whenever we check whether all the clauses are satisfied, we will either restrict all tokens of $A \cup B$ to be in A, or restrict all to be in B. Whenever we have to choose a new set of true variables for t odd, we move all tokens from A to B (or the other way around). This movement takes exactly k steps. The same holds for even t and the sets C and D.

Clause checking gadget. The clause checking gadget exists of four parts, called AC, BC, BD and AD, named after which pair of sets we want the variable tokens to be in. All the vertices of the clause checking gadget form a token set, and we refer to the corresponding token as the *clause token*. The token will traverse the gadget parts in the order $AC \to BC \to BD \to AD \to AC \to \dots$. If the token is on AC, we require the variable tokens to be in A and C and we then check whether the clauses hold for the given choice of variables. The other parts are constructed likewise. For an example we refer to Figure 2.

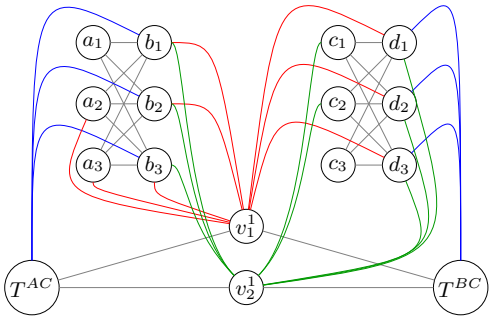


Figure 2 Sketch of part of the construction of Theorem 14. Given are the two variable gadgets for $A^i \cup B^i$ and $C^i \cup D^i$, the AC part of the clause checking gadget with one clause: $C_1 = (x_1^{(t)}, x_3^{(t+1)})$, where t is odd. Hence v_1^1 checks whether a_1 is set to true and v_2^1 checks whether c_3 is set to true.

We now give the construction of this gadget. We create a vertex, T^{AC} that is connected to all $b \in B$. This ensures that if the clause token is on T^{AC}, all tokens from $A \cup B$ are on vertices in A, yet tokens will be able to move from vertices in D to vertices in C.

Suppose $F = C_1 \wedge \dots \wedge C_S$ with each C_i a disjunction of literals. Let $s \in [1, S]$ and let $C_s = y_1 \vee \dots \vee y_{H_s}$ be the sth clause. We create a vertex v_h^s for all $h \in [1, H_s]$. All v_h^s are connected to all vertices in B and D, which ensures that whenever the clause token is on some v_h^s, all variable tokens to be on vertices in A and C and prohibits these variable tokens to move.

Let $h \in [1, H_s]$ and let $j \in [1, q]$ be such that y_h is the jth variable. We ensure that the clause token can only be on v_h^s if the corresponding $x_j^{(t)}$ is modelled as true, that is, the corresponding variable token is on the vertex a_j or c_j (depending on the parity of t). To ensure this, we connect v_h^s to all variables in $A^i \setminus \{a_j\}$ if t is odd and to all variables in $C^i \setminus \{c_j\}$ if t is even, where $i \in [1, k]$ satisfies $j \in P^i$.

We add edges such that $(\{v_h^s\}_{h \in [1, H_s]}, \{v_h^{s+1}\}_{h \in [1, H_{s+1}]})$ forms a complete bipartite graph for all $s \in [1, S-1]$. We connect T^{AC} to all v_h^1 and we connect all v_h^S to T^{BC}, the first vertex of the next gadget. Whenever we move the clause token from T^{AC} to T^{BC}, we have to traverse a vertex v_h^s for each clause C_s, which ensures that the literal y_h in the clause C_s is set to true according to the variable tokens.

The gadget parts for BC, BD and AD are constructed likewise. We omit the details.

Connecting the time counter gadget. We now describe how to connect the vertices in the time counter gadget to those in the clause checking gadget. In the first timer, we create the following edges for $z \in [0, 4q-1]$:

$T^{AC}t_z^1$ when $z \equiv 2$ or $z \equiv 3 \bmod 4$,
$T^{BC}t_z^1$ when $z \equiv 3$ or $z \equiv 0 \bmod 4$,
$T^{BD}t_z^1$ when $z \equiv 0$ or $z \equiv 1 \bmod 4$,
$T^{AD}t_z^1$ when $z \equiv 1$ or $z \equiv 2 \bmod 4$.

This ensures that we can only move the first time token from t_0^1 to t_1^1 if the clause token is on T^{AC}, and that we cannot put the clause token on T^{BC} before having moved the time token. To enforce that the time token moves when the clause token is at T^{AC}, we add edges between any v_h^s vertex in this path and all t_z^1 with $z \not\equiv 1 \mod 4$. The edges are created in a similar manner for the paths following T^{BC}, T^{BD} and T^{AD}.

When the first time token has made q steps, we allow the second time token to move 1 step forward. For $i \in [2,k]$ we add all edges $t_z^i t_y^{i+1}$, *except* for the following $y, z \in [0, 4q-1]$:

$y \equiv 0 \mod 4$ and $z \in [0,q]$,
$y \equiv 1 \mod 4$ and $z \in [q,2q]$,
$y \equiv 2 \mod 4$ and $z \in [2q,3q]$,
$y \equiv 3 \mod 4$ and $z \in [3q,4q-1] \cup \{0\}$.

This ensures, for example, that the $(i+1)$th gadget token can move from t_0^{i+1} to t_1^{i+1} if and only if the ith time gadget token is on t_q^i.

Finally, we add two sets V_{init} and V_{fin} of $2k$ vertices, and add the first set to the initial independent set I_{init} and the second to the final independent set I_{fin}. Each vertex of V_{init} is added to the token set of $A^i \cup B^i$ or $C^i \cup D^i$ for some $i \in [1,2k]$, adding exactly one vertex to each token set, and similarly for V_{fin}.

We create edges uv for all $u \in V_{\mathsf{init}} \cup V_{\mathsf{fin}}$ and v in the clause checking gadget. We also create two vertices c_{init} and c_{fin} that are added to the initial and final independent set respectively, and to the token set of the clause token. We make c_{init} adjacent to T^{AC} and c_{fin} to a vertex T^{XY}, where X, Y depend on the value of r modulo 4.

The vertices c_{init} and c_{fin} are adjacent to all vertices in the time gadgets except for those representing the time 0 and r respectively. The initial independent set also contains the vertices in the time gadget that represent $t = 0$ and similarly I_{fin} contains the vertices that represent r.

Bounding the sequence length. We give a bound ℓ on the length of the reconfiguration sequence, to ensure that only the required moves are made. Before moving the time token, we first move the $2k$ variable tokens into position. We can then move the clause token to T^{AC}, move the first time token so that the time represents 1 and after that take $S+1$ steps to reach T^{BC} (with S the number of clauses in F), at which point we can move the first time token one step forward, and we need to move k variable tokens from A to B. Because we check exactly $r-1$ assignments, we need to move the ith time counter token exactly $\lfloor (r-1)/q^{k-(i-1)} \rfloor$ times. As a last set of moves, we need to move the variable tokens to the set V_{fin}, and the clause token to c_{fin} taking another $2k+1$ steps. Hence, we set the maximum length of the sequence ℓ (from the input of our instance of Binary Timed Partitioned TS-Independent Set Reconfiguration) to

$$4k + 2 + (r-1)(S+k+1) + \sum_{i=1}^{k} \lfloor (r-1)/q^{k-(i-1)} \rfloor.$$

We claim that there is a satisfying assignment for our instance of Long Partitioned Positive Chain Satisfiability if and only if there is a reconfiguration sequence from I_{init} to I_{fin} of length at most ℓ. It is not too difficult to see that a satisfying assignment leads to a reconfiguration sequence (by moving the variable tokens such that they represent the chosen true variables $x_j^{(t)}$ when the time tokens represent time t).

Vice versa, suppose that there is a reconfiguration sequence of length ℓ. This is only possible if the sequence takes a particular form: we need to move the time tokens for $\sum_{i=1}^{k} \lfloor (r-1)/q^{k-(i-1)} \rfloor$ steps, and can only do this if we can move the clause token $(r-1)(S+1)+2$ steps. The moves of the clause token forces us to move k variable tokens between A and B and between C and D a total of $(r-1)$ times, and we need a further $2k$ moves to get these from V_{init} and to V_{fin}. In particular, there is no room for moving a variable token from one position in A to another position in A, without the "time" having moved 4 places. Therefore, for each $i \in [1,k]$ and $t \in [1,r]$, there is a unique j for which we find a variable token on $a_j \in A^i$, $b_j \in B^i$, $c_j \in C^i$ or $d_j \in D^i$ (which letter a, b, c or d we search for depends on the value of t modulo 4) when the time tokens represent time t. This is the variable $x_j^{(t)}$ that we set to true from the tth variable set in partition P^i. ◀

5 Proof of Theorem 4: W[1]-membership

We formulate Timed TJ-Independent Set Reconfiguration with the number of tokens and length of the reconfiguration sequence as combined parameter as an instance of Weighted 3-CNF-Satisfiability.

Weighted 3-CNF-Satisfiability
Given: Boolean formula F on n variables in conjunctive normal form such that each clause contains at most 3 literals; integer K.
Parameter: K.
Question: Can we satisfy F by setting exactly K variables to true?

This proves Theorem 4 since Weighted 3-CNF-Satisfiability is W[1]-complete [4]. We explain how to adjust it to W[2]-membership for the dominating set variant in Appendix C of the full version [2]; the main idea of our proof can be applied for several other reconfiguration problems (all that is needed is that the property of the solution set can be expressed as a CNF formula).

Proof of Theorem 4. Let $(G=(V,E), I_{\mathsf{init}}, I_{\mathsf{fin}}, k, \ell)$ be an instance of Timed TJ-Independent Set Reconfiguration. We set $C = (k+1+\ell)^2$ and $K = \ell(C+1)+(\ell+1)k$. We add the following variables to our Weighted 3-CNF-Satisfiability instance for all $t \in [0,\ell]$:

- $s_{t,v}$, for each vertex $v \in V$. This should be set to true if and only if v has a token at time t.
- $m^{(i)}_{t,v,w}$, for each pair of distinct vertices $v, w \in V$ and for all $i \in [1,C]$. This should be set to true if and only if we move a token from v to w from time $t-1$ to time t.
- $m^{(i)}_{t,\emptyset}$, for all $i \in [1,C]$. This is set to true if no token is moved at from time $t-1$ to time t.
- $a_{t,v}$ for all $v \in V$. This is set to true if and only if v received a token from time $t-1$ to time t.
- $a_{t,\emptyset}$. This is set to true if no vertex received a token from time $t-1$ to time t.

We add clauses that are satisfied if and only if the set of true variables corresponds to a correct TJ-reconfiguration sequence from I_{init} to I_{fin}.

- We have clauses with one literal that ensure that at time 0, we have the initial configuration: for each $v \in I_{\mathsf{init}}$, we have a clause $s_{0,v}$ and for each $v \notin I_{\mathsf{init}}$, we have a clause $\neg s_{0,v}$.
- Similarly, we have clauses that ensure that at time ℓ, we have the final configuration: for each $v \in I_{\mathsf{fin}}$, we have a clause $s_{\ell,v}$ and for each $v \notin I_{\mathsf{fin}}$, we have a clause $\neg s_{\ell,v}$.

- All $m^{(i)}_{t,\star}$ are equivalent: for all distinct $i, j \in [1, C]$, for all $t \in [1, \ell]$ and for all distinct $v, w \in V$, we add the clauses $\neg m^{(i)}_{t,v,w} \vee m^{(j)}_{t,v,w}$ and $m^{(i)}_{t,v,w} \vee \neg m^{(j)}_{t,v,w}$. For all distinct $i, j \in [1, C]$, for all $t \in [1, \ell]$, we add the clauses $\neg m^{(i)}_{t,\emptyset} \vee m^{(j)}_{t,\emptyset}$ and $m^{(i)}_{t,\emptyset} \vee \neg m^{(j)}_{t,\emptyset}$.
- We have clauses that ensure that at each time $t \in [1, \ell]$, at most one move is selected: for any two distinct pairs of distinct vertices (v, w) and (v', w'), we add the clauses $\neg m^{(1)}_{t,v,w} \vee \neg m^{(1)}_{t,v',w'}$ and $\neg m^{(1)}_{t,v,w} \vee \neg m^{(1)}_{t,\emptyset}$.
- For $t \in [1, \ell]$, if the move $m^{(1)}_{t,v,w}$ is selected, then v lost a token and w obtained a token from time $t-1$ to time t: $\neg m^{(1)}_{t,v,w} \vee s_{t-1,v}$, $\neg m^{(1)}_{t,v,w} \vee \neg s_{t-1,w}$, $\neg m^{(1)}_{t,v,w} \vee \neg s_{t,v}$ and $\neg m^{(1)}_{t,v,w} \vee s_{t,w}$.
- For $t \in [1, \ell]$, tokens on vertices not involved in the move remain in place. For all distinct $v, w, u \in V$, we add the clauses

  $\neg m^{(1)}_{t,\emptyset} \vee \neg s_{t-1,v} \vee s_{t,v}$,
  $\neg m^{(1)}_{t,\emptyset} \vee s_{t-1,v} \vee \neg s_{t,v}$,
  $\neg m^{(1)}_{t,v,w} \vee \neg s_{t-1,u} \vee s_{t,u}$ and
  $\neg m^{(1)}_{t,v,w} \vee s_{t-1,u} \vee \neg s_{t,u}$.

- We record if a token was added to a vertex: for all $t \in [1, \ell]$ and $v \in V$, we add the clause $s_{t-1,v} \vee \neg s_{t,v} \vee a_{t,v}$. This in particular ensures that $a_{t,v}$ is true when $m^{(1)}_{t,v,w}$ is true for some vertex $w \neq v$.
- No move happened if and only if no token was added: for all $t \in [1, \ell]$ we add the clauses $\neg m^{(1)}_{t,\emptyset} \vee a_{t,\emptyset}$ and $\neg a_{t,\emptyset} \vee m^{(1)}_{t,\emptyset}$.
- At most one $a_{t,\star}$ is set to true, implying that at most one taken gets added at each time step: for all $t \in [1, \ell]$ and distinct $v, w \in V$, we add the clauses $\neg a_{t,v} \vee \neg a_{t,w}$ and $\neg a_{t,v} \vee \neg a_{t,\emptyset}$.
- Finally, we check whether the current set forms an independent set: for all edges $vw \in E(G)$ and $t \in [0, \ell]$, we add the clause $\neg s_{t,v} \vee \neg s_{t,w}$.

If there is a TJ-independent set reconfiguration sequence $I_{\mathsf{init}} = I_0, \ldots, I_T = I_{\mathsf{fin}}$ with $T \leq \ell$, then we set $s_{t,v}$ to true if and only if $v \in I_t$ for $t \in [0, T]$. For all $t \in [T, \ell]$, we set $s_{t,v}$ to true if and only if $v \in I_T$.

Let $t \in [1, \ell]$. If $I_t = I_{t-1}$, we set $a_{t,\emptyset}$ to true and $m^{(i)}_{t,\emptyset}$ to true for all $i \in [1, C]$. Otherwise, we find $I_t = I_{t-1} \setminus \{v\} \cup \{w\}$ for some $v, w \in V$ and we set $m^{(i)}_{t,v,w}$ and $a_{t,v}$ to true for all $i \in [1, C]$. All other $m^{(i)}_{t,\star}$ are set to false. This gives a satisfying assignment with exactly $\ell(C+1) + (\ell+1)k = K$ variables set to true.

Suppose now that there is a satisfying assignment with K variables set to true. At most one $a_{t,v}$ variable can be true for each $t \in [1, \ell]$. Exactly k variables of the form $s_{0,v}$ are set to true by the initial condition. If there are k' tokens true at time t, then there are at most $k'+1$ tokens true at time $t+1$ and so the $s_{t,v}$ and $a_{t,v}$ variables together can constitute at most $((k+\ell)+1)\ell \leq C-1$ true variables. Therefore, there must be strictly more than $(C+1)(\ell-1)$ variables of the form $m^{(i)}_{t,\star}$ that are set to true. Since $m^{(i)}_{t,\star}$ must take the same value as $m^{(j)}_{t,\star}$, there must be at least ℓ variables of the form $m^{(1)}_{t,\star}$ that are set to true. There can be at most one per time step t, and so there is exactly one per time step. We consider the TJ-independent set reconfiguration sequence $I_{\mathsf{init}} = I'_0, \ldots, I'_\ell = I_{\mathsf{fin}}$ where for $t \in [1, \ell]$ we define $I'_t = I'_{t-1}$ if $m_{t,\emptyset}$ is true, and $I'_t = I'_{t-1} \setminus \{v\} \cup \{w\}$ if $m^{(1)}_{t,v,w}$ is true. The subsequence $I_{\mathsf{init}} = I_0, \ldots, I_T = I_{\mathsf{fin}}$ obtained by removing I'_t if $I'_t = I'_{t-1}$, is now a valid TJ-independent set reconfiguration sequence. ◀

6 Conclusion

We showed that for independent set reconfiguration problems parameterized by the number of tokens, the complexity may vary widely depending on the way the length ℓ of the sequence is treated. If no bound is given, then we ask for the existence of an undirected path in the reconfiguration graph[5] and indeed the problem is XL-complete. If ℓ is given in binary, then we may in particular choose it larger than the maximum number of vertices in the reconfiguration graph, and so this problem is at least as hard as the previous. We show it to be XNL-complete. When ℓ is given in unary, it is easier to have a running time polynomial in ℓ, and indeed the problems becomes XNLP-complete. When ℓ is taken as parameter, the problem is $W[1]$-complete.

On the other hand, switching the rules of how the tokens may move does not affect the parameterized complexity, and the results for dominating set reconfiguration are also similar. It would be interesting to investigate for which graph classes switching between token jumping and token sliding does affect the parameterized complexities. We give an explicit suggestion below.

▶ **Problem 15.** *For which graphs H is TJ-Independent Set Reconfiguration equivalent to TS-Independent Set Reconfiguration under pl-reductions for the class of graphs with no induced H?*

The answer might also differ for Independent Set Reconfiguration and Dominating Set Reconfiguration. We remark that TJ-Clique Reconfiguration and TS-Clique Reconfiguration have the same complexity for all graph classes [8].

References

1. Hans L. Bodlaender, Carla Groenland, Jesper Nederlof, and Céline M. F. Swennenhuis. Parameterized Problems Complete for Nondeterministic FPT time and Logarithmic Space. *arXiv:2105.14882*, 2021.
2. Hans L. Bodlaender, Carla Groenland, and Céline M. F. Swennenhuis. Parameterized complexities of dominating and independent set reconfiguration. *CoRR*, abs/2106.15907, 2021. `arXiv:2106.15907`.
3. Yijia Chen, Jörg Flum, and Martin Grohe. Bounded nondeterminism and alternation in parameterized complexity theory. In *18th Annual IEEE Conference on Computational Complexity (Complexity 2003), 7-10 July 2003, Aarhus, Denmark*, pages 13–29. IEEE Computer Society, 2003. `doi:10.1109/CCC.2003.1214407`.
4. Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer, 1999.
5. Michael Elberfeld, Christoph Stockhusen, and Till Tantau. On the space and circuit complexity of parameterized problems: Classes and completeness. *Algorithmica*, 71(3):661–701, 2015. `doi:10.1007/s00453-014-9944-y`.
6. 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. *Theoretical Computer Science*, 412(12):1054–1065, 2011.
7. Takehiro Ito, Marcin Kamiński, Hirotaka Ono, Akira Suzuki, Ryuhei Uehara, and Katsuhisa Yamanaka. On the parameterized complexity for token jumping on graphs. In T. V. Gopal, Manindra Agrawal, Angsheng Li, and S. Barry Cooper, editors, *Theory and Applications of Models of Computation*, pages 341–351, Cham, 2014. Springer International Publishing.

[5] The reconfiguration graph has the possible token configurations as vertex set, and there is an edge between two configurations if we can go from one to the other with a single move.

8 Takehiro Ito, Hirotaka Ono, and Yota Otachi. Reconfiguration of cliques in a graph. In *Proceedings of the 12th Annual Conference on Theory and Applications of Models of Computation*, pages 212–223, 2015.

9 Marcin Kamiński, Paul Medvedev, and Martin Milanič. Complexity of independent set reconfigurability problems. *Theoretical Computer Science*, 439:9–15, 2012. doi:10.1016/j.tcs.2012.03.004.

10 Harry R. Lewis and Christos H. Papadimitriou. Symmetric space-bounded computation. *Theoretical Computer Science*, 19(2):161–187, 1982.

11 Daniel Lokshtanov and Amer E. Mouawad. The complexity of independent set reconfiguration on bipartite graphs. *ACM Trans. Algorithms*, 15(1), 2018.

12 Amer E. Mouawad, Naomi Nishimura, Venkatesh Raman, Narges Simjour, and Akira Suzuki. On the Parameterized Complexity of Reconfiguration Problems. *Algorithmica*, 78(1):274–297, 2017. doi:10.1007/s00453-016-0159-2.

13 Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4), 2018.

14 Jan van den Heuvel. The complexity of change. *Surveys in Combinatorics*, 409:127–160, 2013.

15 Michael Wehar. *On the complexity of intersection non-emptiness problems*. PhD thesis, State University of New York at Buffalo, 2017.

Twin-Width and Polynomial Kernels

Édouard Bonnet
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Eun Jung Kim
Université Paris-Dauphine, PSL University, CNRS UMR7243, LAMSADE, Paris, France

Amadeus Reinald
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Stéphan Thomassé
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Rémi Watrigant
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Abstract

We study the existence of polynomial kernels for parameterized problems without a polynomial kernel on general graphs, when restricted to graphs of bounded twin-width. It was previously observed in [Bonnet et al., ICALP'21] that the problem k-Independent Set allows no polynomial kernel on graph of bounded twin-width by a very simple argument, which extends to several other problems such as k-Independent Dominating Set, k-Path, k-Induced Path, k-Induced Matching. In this work, we examine the k-Dominating Set and variants of k-Vertex Cover for the existence of polynomial kernels.

As a main result, we show that k-Dominating Set does not admit a polynomial kernel on graphs of twin-width at most 4 under a standard complexity-theoretic assumption. The reduction is intricate, especially due to the effort to bring the twin-width down to 4, and it can be tweaked to work for Connected k-Dominating Set and Total k-Dominating Set with a slightly worse bound on the twin-width.

On the positive side, we obtain a simple quadratic vertex kernel for Connected k-Vertex Cover and Capacitated k-Vertex Cover on graphs of bounded twin-width. These kernels rely on that graphs of bounded twin-width have Vapnik-Chervonenkis (VC) density 1, that is, for any vertex set X, the number of distinct neighborhoods in X is at most $c \cdot |X|$, where c is a constant depending only on the twin-width. Interestingly the kernel applies to any graph class of VC density 1, and does not require a witness sequence. We also present a more intricate $O(k^{1.5})$ vertex kernel for Connected k-Vertex Cover.

Finally we show that deciding if a graph has twin-width at most 1 can be done in polynomial time, and observe that most graph optimization/decision problems can be solved in polynomial time on graphs of twin-width at most 1.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Fixed parameter tractability

Keywords and phrases Twin-width, kernelization, lower bounds, Dominating Set

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.10

Related Version *Full Version*: https://arxiv.org/abs/2107.02882

Funding The first, fourth, and fifth authors were supported by the ANR project DIGRAPHS (ANR-19-CE48-0013-01), the first, second, fourth, and fifth authors were supported by the ANR project TWIN-WIDTH (ANR-21-CE48-0014-01), and the second author was supported by the ANR project ASSK (ANR-18-CE40-0025-01) from French National Research Agency.

Acknowledgements We thank Noga Alon and Bart M. P. Jansen for independently asking whether k-Dominating Set admits a polynomial kernel on classes of bounded twin-width, an interesting question that led to our main result.

1 Introduction

The *twin-width* of a graph can be defined in the following way. A *partition sequence* of an n-vertex graph G, is a sequence $\mathcal{P}_n, \ldots, \mathcal{P}_1$ of partitions of its vertex set $V(G)$, such that $\mathcal{P}_n$ is the set of singletons $\{\{v\} \ : \ v \in V(G)\}$, $\mathcal{P}_1$ is the singleton set $\{V(G)\}$, and for every $2 \leqslant i \leqslant n$, $\mathcal{P}_{i-1}$ is obtained from $\mathcal{P}_i$ by merging two of its parts into one. Two parts P, P' of a same partition $\mathcal{P}$ of $V(G)$ are said *homogeneous* if either every pair of vertices $u \in P, v \in P'$ are non-adjacent, or every pair of vertices $u \in P, v \in P'$ are adjacent. Finally the twin-width of G is the least integer d such that there is partition sequence $\mathcal{P}_n, \ldots, \mathcal{P}_1$ of G with every part of every $\mathcal{P}_i$ $(1 \leqslant i \leqslant n)$ being homogeneous to every other parts of $\mathcal{P}_i$ but at most d. We call such a partition sequence a *d-sequence*.

On the one hand, a surprisingly wide variety of graphs have low twin-width. Graph classes with bounded twin-width include classes with bounded treewidth, or even rank-width, proper minor-closed classes, every hereditary proper subclass of permutation graphs, bounded-degree string graphs [6], classes with bounded queue or stack number, some expander families [4]. Furthermore on those particular classes, we can find (non necessarily optimum) $O(1)$-sequences in polynomial time. We observe that such an approximation algorithm is still missing in general graphs, but exists for *ordered* binary structures [5].

On the other hand, bounded twin-width classes have interesting algorithmic and structural properties. Remarkably, given a partition sequence witnessing that an n-vertex graph G has twin-width at most d, and a first-order sentence φ, one can decide if φ holds in G in time $f(|\varphi|, d)\, n$ for a computable, but non-elementary, function f [6]. That general framework is called *first-order model checking*, and generalizes problems like k-Independent Set with $\varphi = \exists x_1 \ldots \exists x_k \bigwedge_{1 \leqslant i < j \leqslant k} \neg(x_i = x_j \vee E(x_i, x_j))$ and k-Dominating Set with $\varphi = \exists x_1 \ldots \exists x_k \forall x \bigvee_{1 \leqslant i \leqslant k} (x = x_i \vee E(x, x_i))$. For these two particular problems, though, a much better running time of $2^{O_d(k)}\, n$ is possible [3]. In contrast, an algorithm running in time $f(k)n^{o(k)}$ for either of these problems on *general* graphs, with f being any computable function, would imply the improbable (or at least breakthrough) result that 3-SAT can be solved in subexponential time [10].

Now we know that k-Independent Set and k-Dominating Set are fixed-parameter tractable (FPT), i.e., solvable in time $f(k)\, n^{O(1)}$, on graphs of bounded twin-width given with an $O(1)$-sequence, one can then ask whether polynomial kernels exist. A *kernel* is a polytime algorithm that produces, given an instance of a parameterized problem Π, an equivalent instance of Π (i.e., the output is a YES-instance if and only if the input is a YES-instance) of size only function of the parameter. A *polynomial kernel* is a kernel for which the latter function is polynomial. Any decidable problem with a kernel is FPT, and any FPT problem admits a kernel. However not every FPT problem is believed to have a polynomial kernel. And indeed such an outcome would imply an unlikely collapse of complexity classes.

We already observed that there is a constant d such that k-Independent Set is highly unlikely to have a polynomial kernel on graphs with twin-width at most d [3]. The OR-composition is straightforward from the following facts: (1) cliques have twin-width 0 and planar graphs have bounded twin-width [6], (2) the twin-width of every graph is the maximum twin-width of its modules and quotient graph (see Lemma 8), and (3) Maximum Independent Set is NP-hard in (subcubic) planar graphs [30]. Then one can blow every vertex of a clique K_t into a distinct graph among t planar Maximum Independent Set-instances. Facts (1) and (2) imply that the constructed graph has bounded twin-width, while the correctness of the OR-composition is easy to check. Incidentally the exact same reduction rules a polynomial kernel out for k-Independent Dominating Set. Furthermore

Minimum Independent Dominating Set is NP-hard in grid graphs [11], and Maximum Independent Set is NP-hard in subdivisions of grid graphs (since these coincide with planar graphs of degree at most 4). Since these graphs have twin-width at most 4 (see Lemma 9), no polynomial kernel is likely to exist for both problems (even when a 4-sequence is given in the input). It should be noted that this simple reduction fails for k-Dominating Set: one can dominate the constructed graph by picking only two vertices (from two distinct instances).

The parameterized complexity (FPT algorithms and kernels) of k-Dominating Set[1] on "sparse"[2] classes has a rich and interesting history. Subexponential FPT algorithms with running time $2^{O(\sqrt{k})}\, n^{O(1)}$ are known in planar graphs [28, 21], bounded-genus graphs and more generally classes excluding a fixed minor [19, 25, 33], and an FPT algorithm with running time $2^{O(k)}\, n$ exists in classes excluding a fixed topological minor [2]. On these classes the mere existence of an FPT algorithm (but not the particular, enhanced running time) is subsumed by an algorithmic meta-theorem of Grohe, Kreutzer, and Siebertz [32] that says that first-order model checking is FPT in any nowhere dense class.[3] More general than nowhere dense classes are bounded-degeneracy graphs, or further, $K_{t,t}$-free classes, i.e., excluding the biclique $K_{t,t}$ as a subgraph. Alon and Gutner [2] give an FPT algorithm in d-degenerate graphs running in time $k^{O(dk)}\, n$. And Philip, Raman, and Sikdar [40] extend the fixed-parameter tractability of k-Dominating Set to any $K_{t,t}$-free class (for a fixed t). Telle and Villanger [43] further show that k-Dominating Set on $K_{t,t}$-free graphs is FPT for the combined parameter $k + t$.

In parallel to these algorithms, the existence of polynomial, or even linear, kernels have been thoroughly investigated. In 2004, Alber, Fellows, and Niedermeier [1] presented a linear kernel for k-Dominating Set on planar graphs that triggered a series of works. Linear kernels are known on planar graphs [1, 9], bounded-genus graphs [27], apex-minor-free graphs [26], but more generally in any class excluding a fixed topological minor [25]. k-Dominating Set admits a polynomial kernel on graphs of girth 5 (that is, excluding the triangle and the biclique $K_{2,2}$ as a subgraph) [42]. A polynomial kernel of size $O(k^{(t+1)^2})$ is obtained for $K_{t,t}$-free graphs [40], the most general "sparse" class. Contrary to the FPT algorithm, a polynomial kernel in the parameter $k + t$ is highly unlikely [20]. More precisely, for any $\varepsilon > 0$, a kernel of size $k^{(t-1)(t-3)-\varepsilon}$ would imply that coNP $\subseteq$ NP/poly [15]. On classes of bounded expansion[4] k-Dominating Set has a linear kernel, while the seemingly closely related Connected k-Dominating Set has no polynomial kernel [23]. The latter result refines a reduction showing the same lower bound on 2-degenerate graphs [17].

Beyond sparse classes, for which most answers turn out positive, the parameterized complexity of k-Dominating Set seems to conceal many surprises, some of which recently unraveled. We already mentioned that k-Dominating Set is FPT on bounded twin-width graphs given with an $O(1)$-sequence. Let us also mention that the same problem is actually W[1]-hard (hence unlikely FPT) on circle graphs [7]. This is somewhat unexpected since Dominating Set is polytime solvable on permutation graphs [24], a large subclass of circle graphs. On the positive side, k-Dominating Set admits a polynomial kernel on so-called *c-closed graphs* [36], a far-reaching dense generalization of bounded d-degenerate graphs.

[1] All the subsequent results also hold for k-Independent Set.
[2] *Sparse* is an overloaded term; here we use it as *not containing arbitrarily large bicliques as subgraphs*.
[3] The definition of nowhere denseness being technical and unnecessary to the current paper, we refer the interested reader to [39]. Let us just mention that bounded-degree graphs, planar graphs, and proper (topological) minor-closed classes are all nowhere dense.
[4] We will not need a definition of *expansion* here. Bounded expansion classes are more general than topological-minor-free classes and less general than nowhere dense classes.

Our results

We are back to wondering whether k-Dominating Set admits a polynomial kernel on graphs given with an $O(1)$-sequence. On the one hand, a polynomial kernel would "fit all the data points" considering that the examples of bounded twin-width classes previously given are either $K_{t,t}$-free (and one concludes with [40]) or are dense classes on which Minimum Dominating Set is polytime solvable, like bounded rank-width graphs [12], and (subclasses of) permutation graphs [24]. On the other hand, the same could be said of k-Independent Set for which we already ruled out such a kernel. Yet we will see in Section 3.1 that the above OR-composition not working for k-Dominating Set is part of a more general obstacle toward establishing its incompressibility. In the same section we lay down our plan to overcome that obstacle and show the following. Recall that the input to k-Dominating Set is a graph G and an integer k, k construed as the parameter, and the task is to decide whether G has a dominating set of size at most k.

▶ **Theorem 1.** *Unless* coNP $\subseteq$ NP*/poly, k-Dominating Set on graphs of twin-width at most 4 does not admit a polynomial kernel, even if a 4-sequence of the graph is given.*

We mentioned that the same statement holds much more directly for k-Independent Set and k-Independent Dominating Set. With analogous arguments, we can add k-Path, k-Induced Path, k-Induced Matching to the list. Local gadget modifications of the proof of Theorem 1 yield the same kernel lower bound for variants of k-Dominating Set such as Connected k-Dominating Set and Total k-Dominating Set, on graphs of bounded twin-width. More work would be necessary to get the lower bound for twin-width at most 4.

On the positive side, Connected k-Vertex Cover[5] and Capacitated k-Vertex Cover[6] admit polynomial kernels on graphs of bounded twin-width, while such kernels are unlikely on general graphs [20]. Interestingly, our kernelization algorithm does not require an $O(1)$-sequence.

▶ **Theorem 2.** *Connected k-Vertex Cover and Capacitated k-Vertex Cover admit a kernel with $O(k^2)$ vertices on any class of bounded twin-width.*

A linear kernel (in the number of vertices) is known for apex-minor-free classes [26] via the generic framework of bidimensionality, and even for topological-minor-free classes [35]. Another powerful meta-theorem by Gajarský et al. [29] says that every problem with the so-called *finite integer index* (intuitively, that its boundaried graphs provide finitely many distinct contexts) has a linear kernel on bounded expansion classes when parameterized by the vertex cover number (and more generally by the size of a smallest vertex subset whose deletion leaves the graph with bounded treedepth). In particular this yields a linear kernel for Connected k-Vertex Cover, further extending the two previous results. Besides Connected k-Vertex Cover has a polynomial kernel on $K_{t,t}$-free graphs [17].

---

[5] The problem Connected k-VC takes as input a graph G and a parameter k, and asks whether G has a vertex cover of size at most k which induces a connected subgraph of G.

[6] Given a graph G and a capacity function $c : V(G) \to N$, a *capacitated vertex cover* X of G is a vertex cover of G which admits a mapping $\rho : E(G) \to X$ assigning to each vertex $x \in X$ no more edges than its capacity, i.e., $|\rho^{-1}(x)| \leqslant c(x)$ for every $x \in X$. The goal of Capacitated k-VC is to decide, given a graph G with a capacity function $c : V(G) \to N$ and an integer k as the parameter, if G admit a capacitated vertex cover X of size at most k.

Theorem 2 is based on the following useful lemma stating that, in graphs of bounded twin-width, the number of distinct neighborhood traces inside a subset of vertices is at most linear in the size of the subset.

▶ **Lemma 3.** *There is a function f such that for every graph G of twin-width d and $X \subseteq V(G)$, the number of distinct neighborhoods in X, $|\{N(v) \cap X \ : \ v \in V(G)\}|$, is at most $f(d)|X|$.*

A more compact rewording, using the language of Vapnik-Chervonenkis parameters, is that the neighborhood set-system of graphs of bounded twin-width has VC density 1. By extension, we will say that a graph class has VC density at most 1, if its neighborhood hypergraphs do. That bounded twin-width classes have VC density 1 is an interesting property, that is shared with classes of bounded expansion. For example it implies a constant-factor approximation for Min Dominating Set (obtained in a rather different manner in [3]) via small ε-nets [8]. Lemma 3 was independently obtained by Wojciech Przybyszewski in his master thesis [41].

For Connected k-Vertex Cover, an improved kernel can be obtained with a more elaborate argument.

▶ **Theorem 4.** *Connected k-Vertex Cover admits a kernel with $O(k^{1.5})$ vertices on classes with VC density at most 1.*

Table 1 Kernelization results for arguably the three main problems without a polynomial kernel in general graphs, but an interesting story in sparse classes. PK stands for *polynomial kernel*, LK for *linear kernel* (in the number of vertices). The indicated lack of a kernel is under the assumption that coNP $\subseteq$ NP/poly. Our new results are in bold (the results without a reference nor in bold are consequences of results in bold).

	k-Dominating Set	Connected k-DS	Connected k-VC
general	W[2]-complete [22]	W[2]-complete [22]	FPT [13], no PK [20]
bounded expansion	LK [23]	FPT [18], no PK [23]	LK [29]
bounded biclique	PK [40]	FPT [43], no PK [17]	PK, no LK [17]
bounded degeneracy	PK [40]	FPT [31], no PK [17]	PK, no LK [17, 15]
$K_{1,3}$-free	PK [34]	FPT, no PK [34]	LK (trivial)
$K_{1,4}$-free	W[2]-complete [16]	W[2]-complete [16]	LK (trivial)
bounded twin-width	FPT [6], no PK	FPT [6], **no PK**	$O(k^{1.5})$-vertex kernel
twin-width at most 4	FPT [6], **no PK**	FPT [6]	$O(k^{1.5})$-vertex kernel
twin-width at most 1	**in P**	**in P**	**in P**
VC density at most 1	no PK	no PK	**$\mathbf{O(k^{1.5})}$-vertex kernel**

Finally we extend cograph recognizability (cographs are exactly the graphs with twin-width 0) and prove:

▶ **Theorem 5.** *One can decide in polynomial time if a graph has twin-width at most 1.*

In case the input graph has indeed twin-width at most 1, a 1-sequence is found in polynomial time. Furthermore we observe that a wide class of graph problems is efficiently solvable on inputs of twin-width at most 1. See Table 1 for a summary of most of our results, together with the relevant pointers on other graph classes.

2 Preliminaries

We make this section light to keep the extended abstract legible; a complete preliminary section can be found in the full version. We denote by $[i, j]$ the set of integers $\{i, i+1, \ldots, j-1, j\}$, and by $[i]$ the set of integers $[1, i]$. If $\mathcal{X}$ is a set of sets, we denote by $\cup\mathcal{X}$ their union. The notation $O_d(\cdot)$ gives an asymptotic behavior when d is seen as a constant.

An injective mapping $\eta : V(H) \to V(G)$ *witnesses* that H is a subgraph of G if $uv \in E(H)$ implies $\eta(u)\eta(v) \in E(G)$. A bijective mapping $\eta : V(H) \to V(G)$ *witnesses* that H is a *spanning* subgraph of G if $uv \in E(H)$ implies $\eta(u)\eta(v) \in E(G)$.

The *strict half-graph of height* t is (up to isomorphism) the graph with vertex set $\{a_1, \ldots, a_t, b_1, \ldots, b_t\}$ and edge set $\{a_i b_j \ : \ i < j, i \in [t], j \in [t]\}$. One can see $\{a_1, \ldots, a_t\}$ *oriented toward* $\{b_1, \ldots, b_t\}$ in their realization of the relation $<$ over the indices. The *ℓ-cycle of strict half-graphs of height t* is (up to isomorphism) the graph with vertex set $\{a_1^p, \ldots, a_t^p \ : \ p \in [0, \ell-1]\}$ and edge set $\{a_i^p a_j^{p+1 \mod \ell} \ : \ i < j, i \in [t], j \in [t], p \in [0, \ell-1]\}$. Informally it is the graph obtained from an ℓ-vertex cycle by replacing every edge by a strict half-graph of height t with a consistent, say, clock-wise orientation. See Figure 2 for an example of a 5-cycle of strict half-graphs of height 6 (realized by the black edges on the rounded black boxes). A *strict half-graph* is, for some natural t, the strict half-graph of height t. A *cycle of strict half-graphs* is, for some natural ℓ, the ℓ-cycle of strict half-graphs of same height.

The $n \times m$ *grid* is the graph with vertex set $[n] \times [m]$ and edges between any pair of vertices $(x, y), (x+1, y)$ or $(x, y), (x, y+1)$. A *grid* is an $n \times m$ grid for some integer n and m. A *grid graph* is an induced subgraph of a grid. To insist that we consider a grid and not a mere grid graph, we may use the term *complete grid*.

The *neighborhood hypergraph* of a graph G has vertex set $V(G)$ and edge set $\{N(v) \ : \ v \in V(G)\}$. A family of hypergraphs $\mathcal{H}$ has *Vapnik-Chervonenkis (VC) density at most 1* if there is a constant c such that for every hypergraph $H \in \mathcal{H}$ and every $X \subseteq V(H)$, $|\{X \cap e \ : \ e \in E(H)\}| \leqslant c \cdot |X|$.

2.1 Contraction sequences and twin-width

A *trigraph* G has vertex set $V(G)$, black edge set $E(G)$, and red edge set $R(G)$, with $E(G)$ and $R(G)$ disjoint. The *total graph* of trigraph G is the graph G' with $V(G') = V(G)$ and $E(G') = E(G) \cup R(G)$. The *subtrigraph* of G induced by S is the trigraph H with $V(H) = S$, $E(H) = E(G) \cap \binom{S}{2}$, and $R(H) = R(G) \cap \binom{S}{2}$. H is then called an *induced subtrigraph* of G.

The *set of neighbors* $N_G(v)$ of a vertex v in a trigraph G consists of all the vertices adjacent to v by a black or red edge. A d-trigraph is a trigraph G such that the *red graph* $(V(G), R(G))$ has degree at most d. In that case, we also say that the trigraph has *red degree* at most d. A *contraction* or *identification* in a trigraph G consists of merging two (non-necessarily adjacent) vertices u and v into a single vertex z, and updating the edges of G in the following way. Every vertex of the symmetric difference $N_G(u) \triangle N_G(v)$ is linked to z by a red edge. Every vertex x of the intersection $N_G(u) \cap N_G(v)$ is linked to z by a black edge if both $ux \in E(G)$ and $vx \in E(G)$, and by a red edge otherwise. The rest of the edges (not incident to u or v) remain unchanged. See Figure 1 for an illustration.

A *d-sequence* (or *contraction sequence*) is a sequence of d-trigraphs $G_n, G_{n-1}, \ldots, G_1$, where $G_n = G$, $G_1 = K_1$ is the graph on a single vertex, and G_{i-1} is obtained from G_i by performing a single contraction of two (non-necessarily adjacent) vertices. We observe that G_i has precisely i vertices, for every $i \in [n]$. The twin-width of G, denoted by $tww(G)$, is the minimum integer d such that G admits a d-sequence. Note that, in what precedes, the

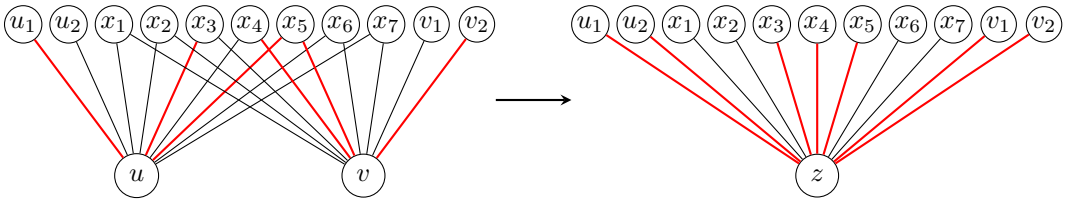


Figure 1 Contraction of vertices u and v, and how the edges of the trigraph are updated.

initial structure $G_n = G$ may be a trigraph instead of a graph. Thus we defined twin-width more generally for trigraphs. Similarly a *partial d-sequence* from a n-vertex trigraph G to an i-vertex trigraph H is a sequence of d-trigraphs $G = G_n, G_{n-1}, \dots, G_i = H$. Observe that if G has a partial d-sequence to H, and H has itself a d-sequence, then the concatenation of these sequences is a d-sequence for G.

Here are useful facts about twin-width, whose proofs are trivial or can be found in the full version.

▶ **Observation 6.** *Let G be a trigraph and H be an induced subtrigraph of G. Then, $tww(H) \leqslant tww(G)$.*

▶ **Observation 7.** *Let G, G' be two trigraphs such that $V(G) = V(G')$, $R(G) \subseteq R(G')$, $E(G') \subseteq E(G)$, and $R(G) \cup E(G) \subseteq R(G') \cup E(G')$. Then $tww(G) \leqslant tww(G')$.*

▶ **Lemma 8.** *Let G be a graph and $\mathcal{H} = \{H_1, H_2, \dots, H_\ell\}$ be its modular partition. Then,*

$$tww(G) = \max\{\max_{i\in[\ell]} tww(H_i), tww(G/\mathcal{H})\}.$$

▶ **Lemma 9.** *Any trigraph whose total graph is a subdivision of a subgraph of a grid has twin-width at most 4.*

2.2 Kernels or lack thereof

For a parameterized problem $\mathcal{Q}$, a *kernel* of size bounded by a function f is a polynomial-time reduction $\rho : \Sigma^* \times N \to \Sigma^* \times N$ such that $(x, k) \in \mathcal{Q}$ if and only if $\rho(x, k) \in \mathcal{Q}$, and $|\rho(x, k)| \leqslant f(k)$. A kernel is said *linear*, *quadratic*, or *polynomial*, if the function f can be chosen linear, quadratic, or polynomial, respectively. We recall the framework of OR-cross-compositions [14], which we will rely on to show the absence of a polynomial kernel in Theorem 1.

▶ **Definition 10.** *A polynomial equivalence relation on Σ^* is an equivalence relation $\mathcal{R}$ when*
(i) *for $x, y \in \Sigma^*$, the equivalence $x\mathcal{R}y$ can be decided in time polynomial in $|x| + |y|$, and*
(ii) *$\mathcal{R}$ restricted to instances of size at most n admits polynomially many equivalence classes.*

We can now formally define an *OR-cross-composition*.

▶ **Definition 11.** *Let $\mathcal{L}$ be a language, $\mathcal{R}$ a polynomial equivalence relation on Σ^* and $\mathcal{Q}$ a parameterized problem. An* OR-cross-composition *from $\mathcal{L}$ to $\mathcal{Q}$ with respect to $\mathcal{R}$ is an algorithm taking as input t $\mathcal{R}$-equivalent instances $x_1, ..., x_t \in \Sigma^*$, running in time polynomial in $\sum_{j=1}^{t} |x_j|$, and outputting an instance $(y, N) \in \Sigma \times N$ such that:*
(i) *N is polynomially bounded in $\max_{j\in[t]} |x_j| + \log t$,*
(ii) *$(y, N) \in \mathcal{Q}$ if and only if there exists some j such that $x_j \in \mathcal{L}$.*

We say that $\mathcal{L}$ cross-composes into $\mathcal{Q}$, and we sometimes refer to output instance (y, N) as the *composed instance*. The following result provides the lower bound under coNP $\subseteq$ NP/poly.

▶ **Theorem 12** ([14]). *If an NP-hard language $\mathcal{L}$ admits an OR-cross-composition into a parameterized problem $\mathcal{Q}$, then $\mathcal{Q}$ does not admit a polynomial kernel unless* coNP $\subseteq$ NP*/poly.*

2.3 Selected results for the short version of the paper

Due to space constraints, most proofs of the results announced in the introduction were omitted, and can be found in the long version of the paper. Section 3 is devoted to our main result: the kernel lower bound for k-Dominating Set in graphs of twin-width at most 4. We provide the intuition behind it, the definition of the particular instances we are reducing from, and finally give the construction as well as a sketch of how we can reach the twin-width bound. In Section 4, we present a sketch of proof that bounded twin-width graphs have VC density 1, and use it to get a kernel with $O(k^{1.5})$ vertices for Connected k-Vertex Cover.

3 Kernel lower bound for k-Dominating Set

3.1 Outline

Let us start explaining why we should *not* expect a simple OR-composition. A standard way to OR-compose t Dominating Set-instances is to have for each instance a "switch", that is, one vertex dominating all but one instance. Then picking the corresponding vertex in the solution, one is left with dominating one chosen instance with a given remaining budget. This is precisely what we want, but how to ensure that one does not activate two switches?

As we previously observed [3], one can use larger weights for the switches. However removing the vertex-weights cannot be done without increasing the twin-width. Another possibility is to force all the budget but one unit (for the switch) within the instances. This requires, say, k vertices called "forcers", each adjacent to a k-th fraction of each instance. Now consider the induced subgraph made by these k vertices, the t switches, and tk vertices of the instances realizing the tk possible neighborhoods toward the former $t + k$ vertices. The two neighborhoods of every pair of vertices in this graph has a large symmetric difference. Thus in particular the overall graph has unbounded twin-width. (Finally known tricks to condense the t switches into $O(\log t)$ vertices do not help, since we want the twin-width to be bounded by an absolute constant.)

So we need a more elaborate way of selecting one instance among t; one, thought primarily to keep the twin-width low. In the previous attempts, the twin-width was increasing too much because of attachments –switches and forcers– external to the instances. We will therefore have instances themselves play these roles. Say that each instance comes with a partition of its vertex set into N parts, each of which containing a vertex solely adjacent to vertices in its part. We place the t instances in a $t \times N$ two-dimensional layout, where each instance occupies a "row," while the j-th part of all the instances form the j-th "column." The switch mechanism is as follows. Every vertex in the j-th part of the i-th instance –say I_i– dominates the $j-1$-st part of the instances with a smaller index, and the $j+1$-st part of the instances with a larger index. In other words, we put a strict half-graph over the parts of two consecutive columns. This is done cylindrically, see Figure 2.

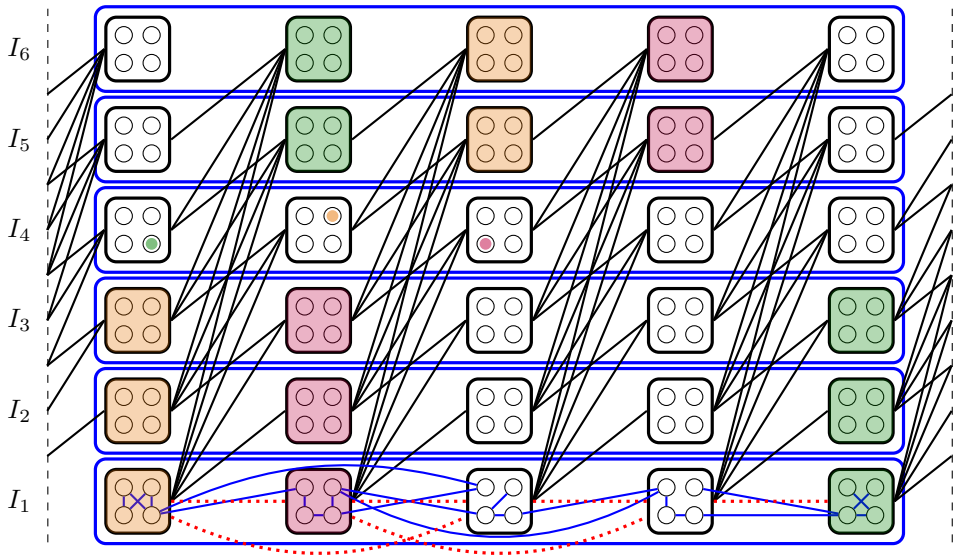


Figure 2 The overall picture. Instances $I_1, \ldots, I_t$ (here with $t = 6$) are in rows, boxed in blue, with their edge also in blue. For the sake of legibility, we only represented the edges of I_1. The red dotted edges are the red edges appearing after contracting every part (boxed in black) into a single vertex. Example of what three vertices picked in the first three parts of I_4 dominates in the other instances. Continuing consistently in I_4 would result in "switching off" all the other instances, while deviating would leave at least one part "white" and not intersected, thus one vertex not dominated.

With that mechanism, a dominating set of a fixed instance I_i (intersecting each of its parts once) is a dominating set of the overall graph. We skip here the details of the reverse direction, but the use of half-graphs and of vertices whose neighborhood in their instance is confined to their own part (the last ingredient is to have a dummy, edgeless top instance I_t) should give a feel for why no other kind of dominating sets of size N can exist.

What about the twin-width bound? Cycles of half-graphs have bounded twin-width. So a natural first step is to contract every part of every instance into a single vertex. Doing so will create some red edges within each row. To ensure that the red degree remains bounded in this first step, a part should be partially adjacent to only a bounded number of other parts. In the second step, we contract the cycle of half-graphs row by row. Thus the red edges of the different instances will progressively stack up. We need to control the accretion with the red edges of each instance mapping onto a common bounded-degree red graph. Finally in the third step, we contract the residual red graph. It should be itself of bounded twin-width, for instance by being planar.

In the next subsection, we show that Minimum Dominating Set remains NP-hard even when inputs are equipped with a vertex-partition satisfying all the properties that we came across in this outline.

3.2 Tailored NP-hardness for Dominating Set

We present a new hardness reduction for Dominating Set. The reduction is designed so as to produce carefully tamed instances, even when compared to existing NP-hardness reductions of Dominating Set (including those on planar instances of bounded degree), and this will be crucial for the subsequent OR-cross-composition, as hinted in the previous section.

The next theorem involves what we will call *snaking grids*. See Figure 3 for an illustration of the 5×10 *snaking grid*, which has $(3 \cdot (5-1)+1)(3 \cdot (10-1)+1)$ vertices. One may observe that the snaking grids are subdivisions of a wall with some extra isolated vertices. We will prefer to think of the snaking grid as a spanning subgraph of a (complete) grid, hence the particular embedding of the figure. The snaking grid is useful as it allows us to superpose a canonical hamiltonian cycle such that the maximum degree remains 3, and thus to bound the twin-width of the composed instance.

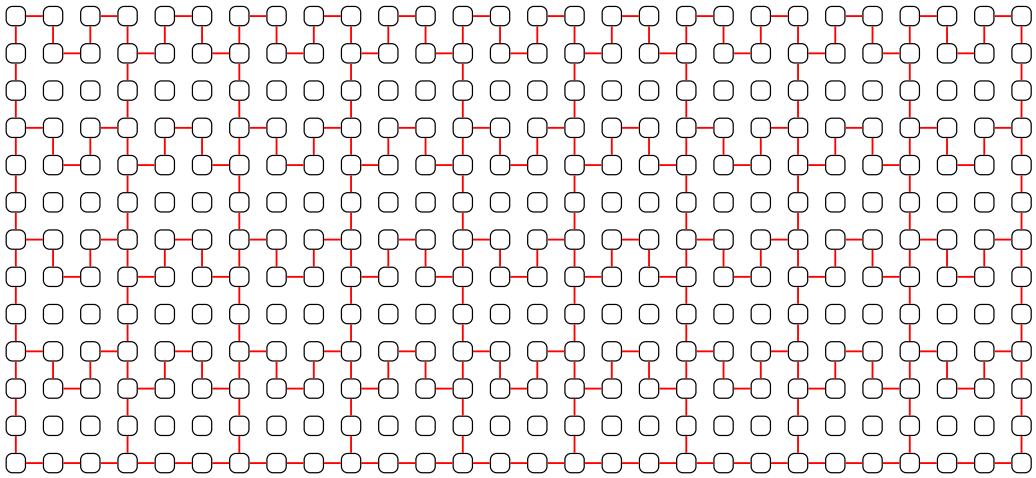


Figure 3 The 5×10 snaking grid.

The following result is obtained through a reduction from the NP-hard problem Planar 3-SAT [37].

▶ **Theorem 13.** *Dominating Set remains* NP*-hard when its input (G, N) comes with a vertex-partition $\mathcal{B} = \{B_1, \ldots, B_N\}$, two positive integers s and t, and a bijective mapping η from $\{B_1, \ldots, B_N\}$ to the vertex set of the $s \times t$ snaking grid such that:*

- **(i)** *G has a partial 4-sequence to the quotient trigraph $G/\mathcal{B}$,*
- **(ii)** *$G/\mathcal{B}$ is a spanning subgraph of the $s \times t$ snaking grid, with t even, witnessed by η, and*
- **(iii)** *every dominating set of G intersects each B_i, for $i \in [N]$.*

3.3 The construction

We now describe the cross-composition from the NP-hard Dominating Set restricted as in Theorem 13 to k-Dominating Set. We use the polynomial equivalence $\mathcal{R}$ to partition all well-formed instances for the restricted k-Dominating Set (those satisfying Theorem 13) with respect to the given parameter N and dimensions (p, q) of the corresponding snaking grid. For any two well-formed instances $(I_i, N_i, \mathcal{B}_i, p_i, q_i, \eta_i), (I_\ell, N_\ell, \mathcal{B}_\ell, p_\ell, q_\ell, \eta_\ell)$, we can check in polynomial time that $N_i = N_\ell$ and $(p_i, q_i) = (p_\ell, q_\ell)$, yielding the polynomial equivalence relation. All ill-formed instances form a single class.

Consider t well-formed instances of the restricted Dominating Set which are taken from an equivalence class with respect to $\mathcal{R}$, and we may consider these instances as $(I_i, N, \mathcal{B}_i, p, q, \eta_i)_{i \in [t]}$. We will construct a k-Dominating Set instance (H, N), with the same parameter, admitting a solution if and only if at least one input instance $(I_i, N, \mathcal{B}_i, p, q, \eta_i)$ admits a solution for the restricted k-Dominating Set. Before composing the input graphs, we introduce a dummy instance in the form of graph I_{t+1} serving to ensure that any valid

(H, N) further admits a solution picking vertices in each column. I_{t+1} is an independent set of size $2N$ on which we partition $V(I_{t+1})$ through $\mathcal{B}_{i+1}$ into N classes of exactly two vertices. Note that since $I_{t+1}/\mathcal{B}_{t+1}$ is an independent set, it is a spanning subgraph of the $p \times q$ snaking grid as witnessed by any bijective η_{t+1} onto the latter.

We first show how to order the partition classes of each instance in the same way with respect to their mapping onto the snaking grid. This ordering will follow a fictitious hamiltonian cycle $(y_1, ..., y_N)$ on the $p \times q$ snaking grid, in the way depicted as the darker red cycles in Figure 4. Referring to the partition of instance $i \in [t+1]$ as $\mathcal{B}_i = \{B_{i,1}, ..., B_{i,N}\}$, we can assume up to the reordering above that $\eta_i(B_{i,j}) = y_j$.

Now, considering all instances over H, a representation of the construction that follows is given in Figure 2. It will be useful to consider the instances in a grid such that $B_{i,j}$ is the cell in the i-th row and j-th column, and we will use the term partition class or cell interchangeably. We can then see instance I_i as row i, and define regular instance columns, omitting the dummy instance, as $C_j = \bigcup_{i \in [t]} B_{i,j}$ for $j \in [N]$.

Construction. We start building our composed graph H as the union of all instances $(I_i)_{i \in [t+1]}$, that is, $V(H) = \bigcup_{i \in [t+1]} V(I_i)$ and $E(I_i) \subseteq E(H)$ for $i \in [t+1]$. Then, our cross-composition proceeds by adding a cycle of strict half-graphs over columns $(C_j)_{j \in [N+1]}$: for $i \in [t+1], j \in [N]$, $B_{i,j}$ forms a biclique with $\bigcup_{i < \ell \leqslant t+1} B_{\ell,j+1}$ (accounting for indices j modulo N). Notice then that the only edges added above lie between columns $C_j, C_{j'}$ with $|j' - j| = 1$, so any edge between two columns differing by at least two indices is an edge of I_i. Each instance class $B_{t+1,j}$ is then adjacent exactly to C_{j-1}. Having ordered the classes of each instance in the same way with respect to their mapping onto the snaking grid, column C_j consists of homologous vertices, all in the same position on their respective grids, see Figure 4. Then, the cycle of half-graphs follows the darker red fictitious hamiltonian cycles mapping to $(y_1, ..., y_N)$. The following lemma allows to conclude the proof of Theorem 1.

▶ **Lemma 14.** *The composed instance has twin-width at most 4, and is positive if and only if one of the input instance is positive.*

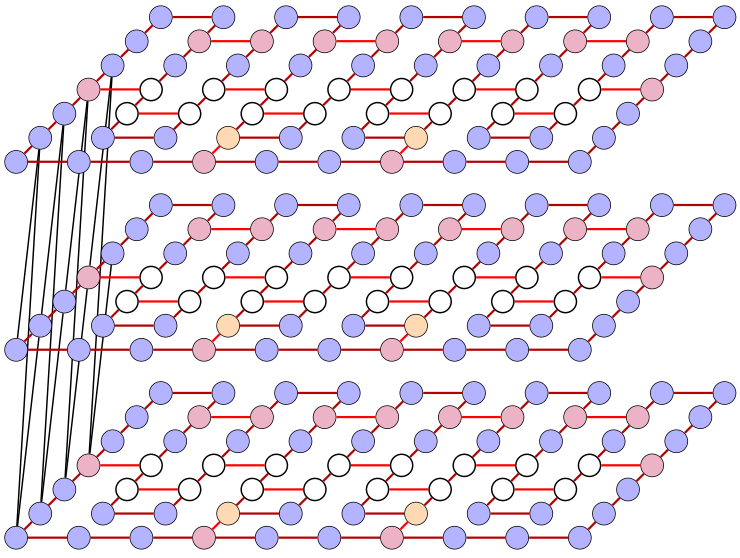


Figure 4 The different layers (instances) linked by the cycle of half-graphs. Only four half-graphs are drawn for the sake of legibility.

How to get twin-width 4, sketch. Let us sketch how to bound the twin-width of the composed graph H. One first contract each $B_{i,j}$ and obtain a trigraph as in Figure 4. It can be shown that the composed graph H admits a partial 4-contraction sequence to $H/\left(\bigcup_{i\in[t+1]}\mathcal{B}_i\right)$. Now, for the purpose of bounding the twin-width of the composed graph, it is useful to note that Observation 7 allows us to add red edges. At this point, the only vertices of large degree stem from the strict half-graphs. We keep those edges black, and we now turn each instance into a red augmented (i.e. additional edges) snaking grid as follows. Since each instance is a spanning subgraph of the $p \times q$ snaking grid, we can first assume that it is a (fully) red snaking grid. Then, the red augmented snaking grid is built by further adding red cycle $(B_{i,1}, ..., B_{i,N})$. By our choice of ordering in the composition, this cycle is the same on every instance with respect to their mapping on the $p \times q$ snaking grid.

We can now describe the contraction of $t+1$ red augmented snaking grids $(I_i)_{i\in[t+1]}$, abusing notation for the now quotiented instances, with the black edges of our composition. We will exhibit a partial 4-contraction sequence eventually contracting every column, now consisting of $t+1$ homologous vertices, that is, all vertices at the same position on their respective snaking grid into a single one. The proof will proceed by induction on the number of augmented snaking grids, our hypothesis at step t being that there exists a partial 4-contraction sequence from t augmented snaking grids to a single one, accounting for the black edges added in the composition. This being true for $t = 1$, assume the result holds for some t and let us consider case $t+1$. We will deal with the two bottommost augmented snaking grids I_1, I_2 in the half-graphs, contracting pairs of homologous vertices, corresponding to the quotiented $(B_{1,j}, B_{2,j})$ thanks to the ordering chosen in the composition.

We argue that the contraction of the first two red snaking grids can be done while bounding the red degree by four. Since the only contracted pairs were homologous, this results in a red augmented snaking grid with no red edges towards grids $i > 2$. The remaining edges of the strict half-graph cycle still form one of height t, which is exactly the induction case for t and achieves to prove the induction. Therefore, there is a partial 4-contraction sequence from our composed graph into a red augmented snaking grid. Then, as the latter is a subgraph of the red complete grid, Lemma 9 yields twin-width at most 4.

4 Polynomial kernels

Let us prove Lemma 3 which, we repeat, is equivalent to saying that the neighborhood hypergraphs of graphs of bounded twin-width have VC density 1. This feature is shared with classes of bounded expansion. Lemma 3 is of independent interest as it opens the door to a common algorithmic treatment for classes of bounded twin-width and of bounded expansion.

We need to introduce some vocabulary on $0,1$-matrices. A *row* (resp. *column*) *division* is a row (resp. column) partition where every part is a consecutive set of rows (resp. columns). A *cell* or *zone* of a matrix M with row and columns divisions $(\mathcal{R}, \mathcal{C})$ is a submatrix $M[R_i, C_j]$ with $R_i \in \mathcal{R}$ and $C_j \in \mathcal{C}$. A *t-division* is a division $(\mathcal{R}, \mathcal{C})$ with $|\mathcal{R}| = |\mathcal{C}| = t$. A matrix is *mixed* if it has at least two distinct rows and at least two distinct columns. A *corner* is a 2×2 contiguous submatrix which is mixed.

▶ **Lemma 3.** *For every graph G of twin-width t and $X \subseteq V(G)$, the number of distinct neighborhoods in X, $|\{N(v) \cap X \ : \ v \in V(G)\}|$, is at most $2^{4c_{2t+2}}|X|$ for some constant c_t depending only on t.*

Proof sketch. We assume for the sake of contradiction that $|\{N(v) \cap X \ : \ v \in V(G)\}| > 2^{4c_{2t+2}}|X|$. For every vertex ordering of G, its adjacency matrix M along this order contains an $|X| \times 2^{4c_{2t+2}}$ submatrix without two equal columns; namely the submatrix of the adjacencies

between X and $2^{4c_{2t+2}}$ vertices with a pairwise distinct neighborhood in X. It can be shown that M has a $2t+2$-mixed minor in the following way. Let $s := |X|$ and create a column division $C_1, \dots, C_s$ into s column parts, each consisting of $2^{4c_{2t+2}}$ consecutive columns. Note that the submatrix of M consisting of the part C_i has rank at least $4c_{2t+2}$ in the binary field F_2, for all $i \in [s]$. Therefore the rows of $M[R, C_i]$ allow a row division into at least $2c_{2t+2}$ parts so that each cell has rank at least 2, thus is mixed, and, by an observation in [6], contains a corner. Let us consider $\mathcal{R}^1$ the row division grouping each pair of rows with indices $2i-1, 2i$, and $\mathcal{R}^2$, grouping each pair of rows $2i, 2i+1$, for $i \in [\lceil s/2 \rceil]$. Observe that one of the two divisions $(\mathcal{R}^1, \{C_i\})$, $(\mathcal{R}^2, \{C_i\})$ of $M[R, C_i]$ contains at least c_{2t+2} zones with a corner, hence mixed.

Without loss of generality, we may assume that at least $\lceil s/2 \rceil$ column parts among $C_1, \dots, C_s$ have at least c_{2t+2} mixed zones when divided by, say, $\mathcal{R}^1$. Consider the column division $\mathcal{C}' = \{C'_1, \dots, C'_{s'}\}$ with $s' \geqslant \lceil s/2 \rceil$, coarsening of $\{C_1, \dots, C_s\}$ such that each part C'_j contains exactly one column part C_i with the property of the previous sentence. Now the number of mixed zones in the division $(\mathcal{R}^1, \mathcal{C}')$ is at least $c_{2t+2} \cdot s'$, and with the correct choice of the constant c_{2t+2}, the celebrated Marcus-Tardos theorem [38] concludes that the division can be further coarsened into $2t+2$-division, where each cell contains a corner. By Grid Minor Theorem of [6], this in turn implies that G has twin-width more than t. ◀

Using the previous lemma, we present here a kernelization algorithm for Connected k-Vertex Cover on bounded twin-width graphs which leads to an instance on $O(k^{1.5})$ vertices. Let X be a vertex cover of G, and let X^b (resp. X^s) be the subsets of X containing all vertices of X with at least $k+1$, respectively at most k, neighbors in $V(G) \setminus X$. Let $Y_1, \dots, Y_q$ be the partition of $V(G) \setminus X$ into maximal modules. For each $i \in [q]$, let X_i be the neighbors of Y_i in X^s. We use one reduction rule, for which the proof of safeness can be found in the full version.

▶ **Reduction Rule 1.** If there is $i \in [q]$ with $X_i \neq \emptyset$ and $|Y_i| \geqslant |X_i| + 2$, delete a vertex of Y_i.

▶ **Proposition 15.** *Connected k-Vertex Cover admits a kernel on $O_t(k^{1.5})$ vertices when the input graphs have twin-width at most t.*

Proof. Let (G, k) be the input instance of Connected k-Vertex Cover. We can safely remove any isolated vertex, and assume that G is connected (otherwise it is a NO-instance). With a 2-approximation algorithm for Vertex Cover, one can find a vertex cover X of G and assume that $|X| \leqslant 2k$. Indeed if this is not the case, we can correctly output a trivial NO-instance because G does not admit a connected vertex cover of size at most k.

Note that Reduction Rule 1 does not disconnect the given graph as we remove a vertex only when it has a twin. Let (G', k) be an instance obtained by exhaustively applying Reduction Rule 1 with the vertex cover X at hand. We classify X into X^b and X^s as before, and $Y_1, \dots, Y_q$ denote the partition of $Y := V(G') \setminus X$ into maximal modules. For each $i \in [q]$, X_i is the neighbors of Y_i in X^s. By Lemma 3, we have $q \leqslant f(t) \cdot 2k$. Because the edge set between X^s and Y is decomposed into the edge sets of complete bipartite graphs on (Y_i, X_i) over $i \in [q]$, the number of edges between X^s and Y is at least

$$\sum_{i=1}^{q} |Y_i| \cdot |X_i| \geqslant \sum_{i=1}^{q} (|Y_i| - 1)^2 \geqslant \frac{1}{q} \cdot \left(\sum_{i=1}^{q} (|Y_i| - 1) \right)^2 \geqslant \frac{1}{q} \cdot (|Y| - q)^2.$$

Suppose that $|Y| - q > 2 \cdot f(t)^{0.5} \cdot k^{1.5}$. Now,

$$\frac{1}{q} \cdot (|Y| - q)^2 > \frac{4 \cdot f(t) \cdot k^3}{2 \cdot f(t) \cdot k} = 2k^2,$$

and hence there are more than $2k^2$ edges between X^s and Y. With $|X^s| \leqslant 2k$, this implies that there exists a vertex in X^s which has more than k neighbors in Y, contradicting the definition of X^s. To conclude, the number of vertices of G' is at most

$$|X| + |Y| \leqslant 2k + 2f(t)k^{1.5} + q \leqslant 2k + 2f(t)k^{1.5} + 2f(t)k = O_t(k^{1.5})$$ ◀

Note that the proof of Proposition 15 only uses the fact that the input graphs have VC density at most 1, so we in fact established Theorem 4.

References

1 Jochen Alber, Michael R. Fellows, and Rolf Niedermeier. Polynomial-time data reduction for dominating set. *J. ACM*, 51(3):363–384, 2004. doi:10.1145/990308.990309.

2 Noga Alon and Shai Gutner. Linear time algorithms for finding a dominating set of fixed size in degenerated graphs. *Algorithmica*, 54(4):544–556, 2009. doi:10.1007/s00453-008-9204-0.

3 Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width III: Max Independent Set and Coloring. *CoRR*, abs/2007.14161, 2020. arXiv:2007.14161.

4 Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width II: small classes. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1977–1996, 2021. doi:10.1137/1.9781611976465.118.

5 Édouard Bonnet, Ugo Giocanti, Patrice Ossona de Mendez, Pierre Simon, Stéphan Thomassé, and Szymon Toruńczyk. Twin-width IV: ordered graphs and matrices. *CoRR*, abs/2102.03117, 2021. arXiv:2102.03117.

6 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 601–612. IEEE, 2020. doi:10.1109/FOCS46700.2020.00062.

7 Nicolas Bousquet, Daniel Gonçalves, George B. Mertzios, Christophe Paul, Ignasi Sau, and Stéphan Thomassé. Parameterized domination in circle graphs. *Theory Comput. Syst.*, 54(1):45–72, 2014. doi:10.1007/s00224-013-9478-8.

8 Timothy M. Chan, Elyot Grant, Jochen Könemann, and Malcolm Sharpe. Weighted capacitated, priority, and geometric set cover via improved quasi-uniform sampling. 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 1576–1585. SIAM, 2012. doi:10.1137/1.9781611973099.125.

9 Jianer Chen, Henning Fernau, Iyad A. Kanj, and Ge Xia. Parametric duality and kernelization: Lower bounds and upper bounds on kernel size. *SIAM J. Comput.*, 37(4):1077–1106, 2007. doi:10.1137/050646354.

10 Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. Strong computational lower bounds via parameterized complexity. *J. Comput. Syst. Sci.*, 72(8):1346–1367, 2006. doi:10.1016/j.jcss.2006.04.007.

11 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-O.

12 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.

13 Marek Cygan. Deterministic parameterized connected vertex cover. In Fedor V. Fomin and Petteri Kaski, editors, *Algorithm Theory - SWAT 2012 - 13th Scandinavian Symposium and Workshops, Helsinki, Finland, July 4-6, 2012. Proceedings*, volume 7357 of *Lecture Notes in Computer Science*, pages 95–106. Springer, 2012. doi:10.1007/978-3-642-31155-0_9.

14 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 4. Springer, 2015.

15 Marek Cygan, Fabrizio Grandoni, and Danny Hermelin. Tight kernel bounds for problems on graphs with small degeneracy. *ACM Trans. Algorithms*, 13(3):43:1–43:22, 2017. doi:10.1145/3108239.

16 Marek Cygan, Geevarghese Philip, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. Dominating set is fixed parameter tractable in claw-free graphs. *Theor. Comput. Sci.*, 412(50):6982–7000, 2011. doi:10.1016/j.tcs.2011.09.010.

17 Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. Kernelization hardness of connectivity problems in d-degenerate graphs. *Discret. Appl. Math.*, 160(15):2131–2141, 2012. doi:10.1016/j.dam.2012.05.016.

18 Anuj Dawar and Stephan Kreutzer. Domination problems in nowhere-dense classes. In Ravi Kannan and K. Narayan Kumar, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009, December 15-17, 2009, IIT Kanpur, India*, volume 4 of *LIPIcs*, pages 157–168. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2009. doi:10.4230/LIPIcs.FSTTCS.2009.2315.

19 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.

20 Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Kernelization Lower Bounds Through Colors and IDs. *ACM Trans. Algorithms*, 11(2):13:1–13:20, 2014. doi:10.1145/2650261.

21 Frederic Dorn. Dynamic programming and fast matrix multiplication. In Yossi Azar and Thomas Erlebach, editors, *Algorithms - ESA 2006, 14th Annual European Symposium, Zurich, Switzerland, September 11-13, 2006, Proceedings*, volume 4168 of *Lecture Notes in Computer Science*, pages 280–291. Springer, 2006. doi:10.1007/11841036_27.

22 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

23 Pål Grønås Drange, Markus Sortland Dregi, Fedor V. Fomin, Stephan Kreutzer, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, Felix Reidl, Fernando Sánchez Villaamil, Saket Saurabh, Sebastian Siebertz, and Somnath Sikdar. Kernelization and sparseness: the case of dominating set. 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 31:1–31:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.STACS.2016.31.

24 Martin Farber and J. Mark Keil. Domination in permutation graphs. *J. Algorithms*, 6(3):309–321, 1985. doi:10.1016/0196-6774(85)90001-X.

25 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Kernels for (connected) dominating set on graphs with excluded topological minors. *ACM Trans. Algorithms*, 14(1):6:1–6:31, 2018. doi:10.1145/3155298.

26 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Bidimensionality and kernels. *SIAM J. Comput.*, 49(6):1397–1422, 2020. doi:10.1137/16M1080264.

27 Fedor V. Fomin and Dimitrios M. Thilikos. Fast parameterized algorithms for graphs on surfaces: Linear kernel and exponential speed-up. In Josep Díaz, Juhani Karhumäki, Arto Lepistö, and Donald Sannella, editors, *Automata, Languages and Programming: 31st International Colloquium, ICALP 2004, Turku, Finland, July 12-16, 2004. Proceedings*, volume 3142 of *Lecture Notes in Computer Science*, pages 581–592. Springer, 2004. doi:10.1007/978-3-540-27836-8_50.

28 Fedor V. Fomin and Dimitrios M. Thilikos. Dominating sets in planar graphs: Branch-width and exponential speed-up. *SIAM J. Comput.*, 36(2):281–309, 2006. doi:10.1137/S0097539702419649.

29 Jakub Gajarský, Petr Hlinený, Jan Obdrzálek, Sebastian Ordyniak, Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. Kernelization using structural parameters on sparse graph classes. *J. Comput. Syst. Sci.*, 84:219–242, 2017. doi:10.1016/j.jcss.2016.09.002.

30 M. R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some simplified np-complete graph problems. *Theor. Comput. Sci.*, 1(3):237–267, 1976. doi:10.1016/0304-3975(76)90059-1.

31 Petr A. Golovach and Yngve Villanger. Parameterized complexity for domination problems on degenerate graphs. In Hajo Broersma, Thomas Erlebach, Tom Friedetzky, and Daniël Paulusma, editors, *Graph-Theoretic Concepts in Computer Science, 34th International Workshop, WG 2008, Durham, UK, June 30 - July 2, 2008. Revised Papers*, volume 5344 of *Lecture Notes in Computer Science*, pages 195–205, 2008. doi:10.1007/978-3-540-92248-3_18.

32 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. *J. ACM*, 64(3):17:1–17:32, 2017. doi:10.1145/3051095.

33 Shai Gutner. Polynomial kernels and faster algorithms for the dominating set problem on graphs with an excluded minor. In Jianer Chen and Fedor V. Fomin, editors, *Parameterized and Exact Computation, 4th International Workshop, IWPEC 2009, Copenhagen, Denmark, September 10-11, 2009, Revised Selected Papers*, volume 5917 of *Lecture Notes in Computer Science*, pages 246–257. Springer, 2009. doi:10.1007/978-3-642-11269-0_20.

34 Danny Hermelin, Matthias Mnich, Erik Jan van Leeuwen, and Gerhard J. Woeginger. Domination when the stars are out. *ACM Trans. Algorithms*, 15(2):25:1–25:90, 2019. doi:10.1145/3301445.

35 Eun Jung Kim, Alexander Langer, Christophe Paul, Felix Reidl, Peter Rossmanith, Ignasi Sau, and Somnath Sikdar. Linear kernels and single-exponential algorithms via protrusion decompositions. *ACM Trans. Algorithms*, 12(2):21:1–21:41, 2016. doi:10.1145/2797140.

36 Tomohiro Koana, Christian Komusiewicz, and Frank Sommer. Exploiting c-closure in kernelization algorithms for graph problems. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 65:1–65:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.65.

37 David Lichtenstein. Planar formulae and their uses. *SIAM J. Comput.*, 11(2):329–343, 1982. doi:10.1137/0211025.

38 Adam Marcus and Gábor Tardos. Excluded permutation matrices and the Stanley-Wilf conjecture. *J. Comb. Theory, Ser. A*, 107(1):153–160, 2004. doi:10.1016/j.jcta.2004.04.002.

39 Jaroslav Nesetril and Patrice Ossona de Mendez. *Sparsity - Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-27875-4.

40 Geevarghese Philip, Venkatesh Raman, and Somnath Sikdar. Polynomial kernels for dominating set in graphs of bounded degeneracy and beyond. *ACM Trans. Algorithms*, 9(1):11:1–11:23, 2012. doi:10.1145/2390176.2390187.

41 Wojciech Przybyszewski and Szymon Toruńczyk. personal communication, 2021.

42 Venkatesh Raman and Saket Saurabh. Short Cycles Make W -hard Problems Hard: FPT Algorithms for W-hard Problems in Graphs with no Short Cycles. *Algorithmica*, 52(2):203–225, 2008. doi:10.1007/s00453-007-9148-9.

43 Jan Arne Telle and Yngve Villanger. FPT algorithms for domination in sparse graphs and beyond. *Theor. Comput. Sci.*, 770:62–68, 2019. doi:10.1016/j.tcs.2018.10.030.

A New Parametrization for Independent Set Reconfiguration and Applications to RNA Kinetics

Laurent Bulteau
LIGM, CNRS, Univ Gustave Eiffel, F77454 Marne-la-vallée, France

Bertrand Marchand
LIX, CNRS UMR 7161, Ecole Polytechnique, Institut Polytechique de Paris, France
LIGM, Marne-la-Valée, France

Yann Ponty
LIX, CNRS UMR 7161, Ecole Polytechnique, Institut Polytechique de Paris, France

Abstract

In this paper, we study the Independent Set (IS) reconfiguration problem in graphs. An IS reconfiguration is a scenario transforming an IS L into another IS R, inserting/removing vertices one step at a time while keeping the cardinalities of intermediate sets greater than a specified threshold. We focus on the *bipartite* variant where only start and end vertices are allowed in intermediate ISs. Our motivation is an application to the *RNA energy barrier* problem from bioinformatics, for which a natural parameter would be the difference between the initial IS size and the threshold.

We first show the para-NP hardness of the problem with respect to this parameter. We then investigate a new parameter, the *cardinality range*, denoted by ρ which captures the maximum deviation of the reconfiguration scenario from optimal sets (formally, ρ is the maximum difference between the cardinalities of an intermediate IS and an optimal IS). We give two different routes to show that this problem is in XP for ρ: The first is a direct $O(n^2)$-space, $O(n^{2\rho+2.5})$-time algorithm based on a separation lemma; The second builds on a parameterized equivalence with the directed pathwidth problem, leading to a $O(n^{\rho+1})$-space, $O(n^{\rho+2})$-time algorithm for the reconfiguration problem through an adaptation of a prior result by Tamaki [20]. This equivalence is an interesting result in its own right, connecting a reconfiguration problem (which is essentially a *connectivity* problem within a *reconfiguration network*) with a *structural* parameter for an auxiliary graph.

We demonstrate the practicality of these algorithms, and the relevance of our introduced parameter, by considering the application of our algorithms on random small-degree instances for our problem. Moreover, we reformulate the computation of the energy barrier between two RNA secondary structures, a classic hard problem in computational biology, as an instance of bipartite reconfiguration. Our results on IS reconfiguration thus yield an XP algorithm in $O(n^{\rho+2})$ for the energy barrier problem, improving upon a partial $O(n^{2\rho+2.5})$ algorithm for the problem.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Applied computing → Bioinformatics

Keywords and phrases reconfiguration problems - parameterized algorithms - RNA bioinformatics - directed pathwidth

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.11

Related Version *Full Version*: `https://hal.inria.fr/hal-03272963`

Supplementary Material *Software*: `https://gitlab.inria.fr/bmarchan/bisr-dpw`

Acknowledgements The authors would like to thank H. Tamaki and Y. Kobayashi for fruitful email exchanges.

1 Introduction

Reconfiguration problems. Reconfiguration problems informally ask whether there exists, between two *configurations* of a system, a *reconfiguration pathway* entirely composed of *legal* intermediate configurations, connected by legal *moves*. In a thoroughly studied sub-category of these problems, configurations correspond to *feasible solutions* of some *optimization problem*, and a feasible solution is legal when its quality is higher than a specified threshold.

Examples of optimization problems for which reconfiguration versions have been studied include Dominating Set, Vertex Cover, Shortest Path or Independent Set, which is our focus in this article. Associated complexities range from polynomial (see [23] for examples) to NP-complete (for bipartite independent set reconfiguration [13]), and even **PSPACE**-complete for many of them [13, 9]. Such computational hardness motivates the study of these problems under the lens of *parametrized complexity* [18, 14, 15, 9], in the hope of identifying tractable sub-regimes. Typical parameters considered by these studies focus on the value of the *quality threshold* (typically a *solution size* bound) defining legal configurations and the length of the reconfiguration sequences.

Directed pathwidth. *Directed pathwidth*, originally defined in [1] and attributed to Robertson, Seymour and Thomas, represents a natural extension of the notions of pathwidth and path decompositions to directed graphs. Like its undirected restriction, it may alternatively be defined in terms of *graph searching* [24], *path decompositions* [4, 6] or *vertex separation number* [11, 20]. An intuitive formulation can be stated as the search for a visit order of the directed graph, using as few active vertices as possible at each step, and such that no vertex may be deactivated until all its in-neighbors have been activated. Although an FPT algorithm is known for the undirected pathwidth [2], it remains open whether computing the directed pathwidth admits a FPT algorithm. XP algorithms [20, 11] are known, and have been implemented in practice [19, 12].

RNA energy barrier. RNAs are single-stranded biomolecules, which fold onto themselves into 2D and 3D structures through the pairing of nucleotides along their sequence [22]. Thermodynamics then favors low-energy structures, and the RNA energy barrier problem asks, given two structures, whether there exists a re-folding pathway connecting them that does not go through unlikely high-energy intermediate states [17, 21]. Interestingly, the problem falls under the wide umbrella of reconfiguration problems described above, namely the reconfiguration of solutions of optimization problems (here, energy minimization). An important specificity of the problem is that the probability of a refolding pathway depends on the energy difference between intermediate states and the *starting point* rather than the absolute energy value. Another aspect of this problem is that, since some pairings of the initial structure may impede the formation of new pairings for the target structure, it induces a notion of *precedence constraints*, and may therefore also be treated as a *scheduling* problem, as carried out in [8, 10].

Problem statement. In our work, we focus on independent set reconfigurations where only vertices from the start or end ISs (L and R) are allowed within intermediate ISs. This amounts to considering the induced subgraph $G[L \cup R]$, bipartite by construction. We write $\alpha(G)$ for the size of a maximum independent set of G (recall that $\alpha(G)$ can be computed in polynomial time on bipartite graphs).

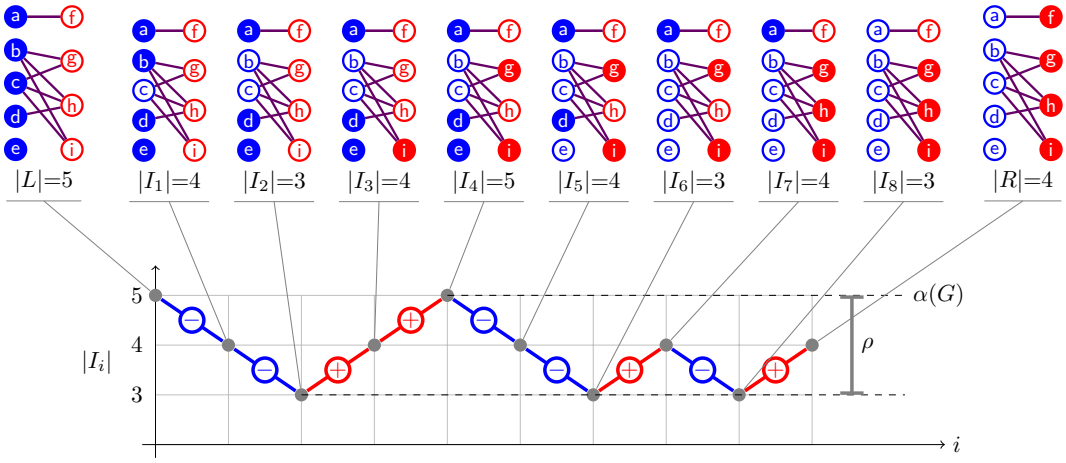


Figure 1 Example of a bipartite independent set reconfiguration from vertices in L (blue) to R (red). Selected vertices at each step have a filled background. All intermediate ISs have size at least 3, and the optimal IS has size 5, so this scenario has a range of 2; it can easily be verified that it is optimal.

Bipartite Independent Set Reconfiguration (BISR)
Input: Bipartite graph $G = (V, E)$ with partition $V = L \cup R$; integer ρ
Parameter: ρ
Output: True if there exists a sequence $I_0 \cdots I_\ell$ of independent sets of G such that
- $I_0 = L$ and $I_\ell = R$;
- $|I_i| \geq \alpha(G) - \rho, \forall i \in [0, \ell]$;
- $|I_i \triangle I_{i+1}| = 1, \forall i \in [0, \ell - 1]$.

False otherwise.

Figure 1 shows an example of an instance of BISR and a possible reconfiguration pathway. We introduce the *cardinality range* (or simply *range*) $\rho = \max_{1 \leq i \leq \ell} \alpha(G) - |I_i|$ as a natural parameter for this problem, since it measures a distance to optimality. As mentioned above, the related parameter in RNA reconfiguration is the *barrier*, denoted k, and defined as $k = \max_{1 \leq i \leq \ell} |L| - |I_i|$. Intuitively, k measures the size difference from the starting point rather than from an "absolute" optimum. Note that $k = \rho - (\alpha(G) - |L|)$, so one has $0 \leq k \leq \rho$. Both parameters are obviously similar for instances where L is close to being a maximum independent set, which is generally the case in RNA applications, but in theory the range ρ can be arbitrarily larger than the barrier k.

Our results. We first prove that in general, the barrier k may not yield any interesting parameterized algorithm, since BISR is Para-NP-hard for this parameter. We thus focus on the range parameter for Bipartite Independent Set Reconfiguration, and prove that it is in XP by providing two distinct algorithmic strategies to tackle it.

Our first algorithmic strategy stems from a parameterized equivalence we draw between BISR and the problem of computing the directed pathwidth of directed graphs. Within this equivalence, the range parameter ρ maps exactly to the directed pathwidth. This allows to apply XP algorithms for Directed Pathwidth to BISR while retaining their complexity, such as the $O(n^{\rho+2})$-time, $O(n^{\rho+1})$-space algorithm from Tamaki [20] (with $n = |V|$). This equivalence between directed pathwidth and bipartite independent set reconfiguration is itself

an interesting result, as it connects a *structural* problem, whose parameterized complexity is open, with a reconfiguration problem of the kind that is routinely studied in parameterized complexity [18, 14, 15, 9].

We also present another more direct algorithm for BISR, with a time complexity of $O(n^{2\rho}\sqrt{n}m)$ (with $m = |E|$) but using only $O(n^2)$ space. It relies on a separation lemma involving, if it exists, a *mixed* maximum independent set of G containing at least one vertex from both parts of the graph. In the specific case of bipartite graphs arising from RNA reconfiguration, we improve the run-time of the subroutine computing a mixed MIS to $O(n^2)$ (rather than $O(\sqrt{n}m)$), with a dynamic programming approach.

We present benchmark results for both algorithms, on random instances of general bipartite graphs as well as instances of the RNA Energy Barrier problem. The approach based on directed pathwidth yields reasonable solving times for RNA strings of length up to ~ 150.

Outline. To start with, Section 2 presents some previously known results related to BISR, as well as some alternative formulations or parameters. Then, Section 3 shows that BISR is in fact equivalent to the computation of *directed pathwidth* in directed graphs. We first present a parameterized reduction from bipartite independent set reconfiguration to an input-restricted version, on graphs allowing for a perfect matching. Then, this version of the problem is shown to be simply equivalent to the computation of directed pathwidth on general directed graphs.

Section 4 presents our direct algorithm for bipartite independent set reconfiguration. More precisely, Section 4.2 presents the separation lemma on which the divide-and-conquer approach of the algorithm is based, while Section 4.3 details the algorithm and its analysis.

To finish, Section 5 explains some optimizations specific to RNA reconfiguration instances, and presents our numerical results.

2 Preliminaries

Previous results. Bipartite Independent Set Reconfiguration was proven NP-complete in [13], through the equivalent k-Vertex Cover Reconfiguration problem. Formulated in terms of RNAs, and restricted to secondary structures (i.e. the subset of bipartite graphs that can be obtained in RNA reconfiguration instances), it was independently proven NP-hard in [17]. To the authors' knowledge, its parameterized complexity remains open.

Independent set reconfiguration in an unrestricted setting (allowing vertices which are outside from the start or end independent sets, i.e. in possibly non-bipartite graphs) when parameterized by the minimum allowed size of intermediate sets has been proven W[1]-hard [18, 9], and fixed-parameter tractable for planar graphs or graphs of bounded degree [14]. Whether this more general problem is in XP for this parameter remains open. We note that in this setting, parameter ρ seems slightly less relevant since it involves computing a maximal independent set in a general graph (i.e. testing if there exists a reconfiguration from $\emptyset$ to $\emptyset$ with range ρ is equivalent to deciding if $\alpha(G) \geq \rho$).

As for algorithms for BISR, the closest precedent is an algorithm by Thachuk et al. [21]. It is restricted to RNA secondary structure conflict graphs, and additionally to conflict graphs for which both parts L and R are *maximum independent sets* of G. In this restricted setting, although it is not stated as such, [21] provides an XP algorithm with respect to the barrier parameter k which then coincides with the range parameter ρ that we introduce.

Restriction to the monotonous case. A reconfiguration pathway for BIPARTITE INDEPENDENT SET RECONFIGURATION is called *monotonous* or *direct* if every vertex is added or removed exactly once in the entire sequence. The length of a monotonous sequence is therefore necessarily: $\ell = |L \cup R| = |L| + |R|$.

Theorem 2 from [13] tells us that if G, ρ is a yes-instance of bipartite independent set reconfiguration, then there exists a *monotonous* reconfiguration between L and R respecting the constraints. We will therefore restrict without loss of generality our study to this simpler case. In the more restricted set studied in [21], this was also independently shown.

Hardness for the barrier parameter. In the general case where L is not necessarily a maximal independent set, the range and barrier parameters (respectively ρ and $k = \rho - (\alpha(G) - |L|)$ may be arbitrarily different. The following result motivates our use of parameter ρ for the parameterized analysis of BISR.

▶ **Proposition 1.** BISR *is* Para-NP*-hard for the energy barrier parameter (i.e.* NP*-hard even with* $k = 0$*).*

Proof. We use additional vertices in R to prove this result. Informally, such a vertex may always be inserted first in a realization: it improves the starting IS from $|L|$ to $|L|+1$, so the lower bound on the rest of the sequence is shifted from $|L| - k$ to $|L| - (k-1)$, effectively reducing the barrier without simplifying the instance. Thus, we build a reduction from the general version of BISR: given a bipartite graph G with parts L and R and an integer ρ, we construct a new instance G' with parts $L' = L$ and R' equal to $R \cup N_R$ and $\rho' = \rho$. N_R is composed of $|L| - (\alpha(G) - \rho)$ isolated vertices (we can assume without loss of generality that this quantity is non-negative, otherwise (G, ρ) is a trivial no-instance), completely disconnected from the rest of the graph.

Note that $\alpha(G') = \alpha(G) + |N_R| = |L| + \rho$, so the barrier in (G', ρ') is $k = \rho - (\alpha(G') - |L|) = 0$. A realization for (G, ρ) can be transformed into a realization for (G', ρ) by inserting vertices from N_R first, and conversely, vertices from N_R can be ignored in a realization for (G', ρ) to obtain a realization for (G, ρ). Therefore, since BISR is NP-Complete, it is also Para-NP-hard w.r.t the barrier k. ◀

Permutation formulation and ρ-realizations. An equivalent representation of a monotonous reconfiguration pathway $I_0 \dots I_\ell$ from L to R for a graph G is a *permutation* P of $L \cup R$. The i-th vertex of the permutation is the vertex that is *processed* (i.e. added or removed) between I_{i-1} and I_i (this formulation lightens the representation of a solution, from a list of vertex sets to a list of vertices). Given a subset X of vertices, we write $\delta(X) = |L \cap X| - |R \cap X|$ and $I(X) = (L \setminus X) \cup (R \cap X) = L \Delta X$ for the set obtained from L after processing vertices from X. Then $|I(X)| = |L| - \delta(X)$. We say that X is *licit* if $I(X)$ is an independent set. For any prefix p of P of length i, we write $V(p)$ (or simply p if the context is clear) for the set of vertices appearing in p, and $I_i = I(V(p))$. A permutation P is licit if $V(p)$ is licit for each prefix p of P; note that P is licit if and only if $\forall r \in R$, the neighborhood $N(r)$ of r in G appears before r in P. Last, we say that P is a *ρ-realization* that is licit and such that for each prefix p, $|I(p)| \geq \alpha(G) - \rho$ (i.e. $\delta(V(p)) \leq \rho + |L| - \alpha(G)$).

3 Connection to Directed Pathwidth

3.1 Definitions

Parameterized reduction. In this section, we provide a definition of directed pathwidth, and then prove its parameterized equivalence to the bipartite independent set reconfiguration problem. We say two problems $\mathcal{P}_1$ and $\mathcal{P}_2$ are parametrically equivalent when there exists both a *parameterized reduction* from $\mathcal{P}_1$ to $\mathcal{P}_2$ and another from $\mathcal{P}_2$ to $\mathcal{P}_1$. A parameterized reduction [5] from problem $\mathcal{P}$ to problem $\mathcal{Q}$ is a function φ from instances of $\mathcal{P}$ to instances of $\mathcal{Q}$ such that (i) $\varphi(x)$ is a yes-instance of $\mathcal{Q} \Leftrightarrow x$ is a yes-instance of $\mathcal{P}$, (ii) $\varphi(x)$ can be computed in time $f(k) \cdot |x|^{O(1)}$, where k is the parameter of x, and (iii) if k is the parameter of x and k' is the parameter of $\varphi(x)$, then $k' \leq g(k)$ for some (computable) function g.

Interval representation. Our definition of directed pathwidth relies on interval embeddings. Alternative definitions can be found, for instance in terms of directed path decomposition or directed vertex separation number [24, 20, 11].

▶ **Definition 2** (Interval representation). *An* interval representation *of a directed graph H associates each vertex $u \in H$ with an interval $I_u = [a_u, b_u]$, with a_u, b_u integers. An interval representation is* valid *when $(u, v) \in E \Rightarrow a_u \leq b_v$. I.e, the interval of u must start before the interval of v ends. If m, M are such that $\forall u, m \leq a_u, b_u \leq M$, we define the* width *of an interval representation as* $\max_{m \leq i \leq M} |\{u | i \in I_u\}|$

▶ **Definition 3** (directed pathwidth). *The* directed pathwidth *of a directed graph H is the minimum possible width of a valid interval representation of H. We note this number $dpw(H)$.*

Nice interval representation. An interval representation is said to be *nice* when no more than one interval bound is associated to any given integer, and the integers associated to interval bounds are exactly $[1 \dots 2 \cdot |V(H)|]$. Any interval representation may be turned into a nice one without changing the width by introducing new positions and "spreading events". See the full version of the article for more details.

Directed graph from perfect matching. Given a bipartite graph G allowing for a perfect matching M, we construct an associated directed graph H in the following way: the vertices of H are the edges of the matching, and $(l, r) \to (l', r')$ is an arc of H iff $(l, r') \in G$. Alternatively, H is obtained from G, M by orienting the edges of G from L to R, and then contracting the edges of M. We will denote this graph $H(G, M)$, and simply call it the *directed graph associated to G, M*. Such a construction is relatively standard and can be found in [7, 25], for instance.

3.2 Directed pathwidth ⇔ Bipartite independent set reconfiguration

Perfect matching case. Our main structural result is the following. Its proof, relying on interval representations, is quite straightforward and can be found in the full version of the article.

▶ **Proposition 4.** *Let G be a bipartite graph allowing for a perfect matching M, and let $H(G, M)$ be the directed graph associated to G, M. Then G allows for a ρ-realization iff $dpw(H(G, M)) \leq \rho$.*
Conversely, given any directed graph H, there exists a bipartite graph G allowing for a perfect matching M such that $H = H(G, M)$ is the directed graph associated to G, M and G allows for a ρ-realization iff $dpw(H) \leq \rho$.

The first half of Proposition 4 is a parameterized reduction from an input-restricted version of BIPARTITE INDEPENDENT SET RECONFIGURATION to directed pathwidth. The restriction is on bipartite graphs allowing for a perfect matching. The second half is a parameterized reduction in the other direction. In both cases, the parameter value is directly transferred, which allows to retain the same complexity when transferring an algorithm from one problem to the other.

Non-perfect-matching case. In the case where G does not allow for a perfect matching, we construct an equivalent instance G' allowing for a perfect matching M', through the addition of new vertices. Specifically, with a bipartite graph G with sides L, R, a maximum matching M of G, and the set U of unmatched vertices in G, we extend G with $|U|$ new vertices in two sets N_L, N_R, giving a new graph G', with sides $L' = L \cup N_L, R' = R \cup N_R$, in the following way ($M'$ is initialized to M):

- For each $u \in L \cap U$, we introduce a new vertex $r(u) \in N_R$, connect it to *all* vertices of L', and add the edge $(u, r(u))$ to M'.
- Likewise, for each $v \in R \cap U$, we introduce $l(v) \in N_L$, connect it to all vertices of R' and add $(v, l(v))$ to M'.

Note that M' is a perfect matching of the extended bipartite graph G'.

▶ **Proposition 5.** *With G, G' defined as above, we have that G allows for a ρ-realization iff G' allows for a ρ-realization.*

Proof. First note that by König's Theorem, $\alpha(G') = |M'| = |M| + |U| = \alpha(G)$, so it suffices to ensure that any realization for G can be transformed into a realization for G' where independent sets are lower-bounded by the same value, and vice versa.

Let P be any ρ-realization of G, then $P' = N_L \cdot P \cdot N_R$ is a ρ-realization for G', with N_L and N_R laid out in any order. Indeed, P' satisfies the precedence constraint, and any intermediate set I in P' satisfies one of the following cases: $L \subseteq I$, $R \subseteq I$, or I is an intermediate set from P, so in any case it has size at least $\alpha(G) - \rho = \alpha(G') - \rho$.

Conversely, because of the all-to-all connectivity between N_L and R and between L and N_R, a realization for G' needs to have N_L before any vertex from R, and have N_R after all vertices from L. Without loss of generality, it is therefore of the form $N_L \cdot P \cdot N_R$ with P a realization of G, and G allows for a ρ-realization. ◀

The construction above in fact yields a parameterized reduction from BIPARTITE INDEPENDENT SET RECONFIGURATION to its input-restricted version on bipartite graphs, allowing for a perfect matching. This input-restricted version is in turn parametrically equivalent to directed pathwidth by Proposition 4. Hence the following corollary:

▶ **Corollary 6.** BIPARTITE INDEPENDENT SET RECONFIGURATION *is parametrically equivalent to* DIRECTED PATHWIDTH

4 An XP algorithm for independent set reconfiguration

4.1 Definitions

We use the permutation representation of reconfiguration scenarios, i.e. licit permutations of vertices. Note that the intersection, as well as the union, of two licit set of vertices are licit. Given a realization P of G and a set of vertices X, we write $P \cap X$ for the sub-sequence of P consisting of the vertices of X, without changing the order. Likewise, $P \setminus X$ denotes the sub-sequence of P consisting of vertices *not* in X.

A *mixed maximum independent set* I of G is an independent set of G of maximum cardinality containing at least a vertex from both parts. Note that not every bipartite graph contains such a set. A *separator* X is a subset of $L \cup R$ such that $I(X)$ is a mixed maximum independent set of G.

4.2 Separation lemma

The separation lemma on which our algorithm is based is proved using the following "modularity" property of the imbalance functions. Interestingly, it is almost the same property (sub-modularity), on a different quantity (the in-degrees of vertices) on which rely the XP algorithm for directed pathwidth [20].

▶ **Lemma 7** (modularity). *The function associating a licit subset to its corresponding independent set $I(X)$ verifies:*

$$|I(X)| + |I(Y)| = |I(X \cup Y)| + |I(X \cap Y)|$$

Proof. We have $I(X) = (L \setminus X) \cup (R \cap X)$. Therefore, $|I(X)| = |L \setminus X| + |R \cap X| = |L| - |L \cap X| + |R \cap X|$. Furthermore, $|(X \cup Y) \cap L| = |(X \cap L) \cup (Y \cap L)| = |X \cap L| + |Y \cap L| - |X \cap Y \cap L|$, and likewise for R. The result stems from a substraction of one equation to the other, and an addition of $|L|$. ◀

Based on this "modularity", the following separation lemma is shown by "re-shuffling" a solution into another one going through a mixed MIS.

▶ **Lemma 8** (separation lemma). *Let X be a separator of G. If P is a ρ-realization for G, then $(P \cap X) \cdot (P \setminus X)$ is also a ρ-realization for G.*

Proof. Let P be a ρ-realization for G and $P' = (P \cap X) \cdot (P \setminus X)$ a reshuffling, where X is processed first.

Consider p' a prefix of P'. There are two cases:

1. p' is included in (or equal to) $P \cap X$. In this case, $\exists p$ prefix of P such that: $p' = p \cap X$. We therefore have $|I(p')| = |I(p)| + |I(X)| - |I(p \cup X)|$, and since $|I(X)|$ is a maximum independent set of G, $|I(p')| \geq |I(p)| \geq \alpha(G) - \rho$.
2. $P \cap X$ is included in p. In that case, $\exists p$ prefix of P such that $p' = p \cup X$. We have, likewise, $|I(p')| = |I(p)| + |I(X)| - |I(p \cap X)|$ and conclude the same way. ◀

The separation allows for a divide-and-conquer approach: if we identify a separator X in G, i.e. a licit subset of G such that $I(X)$ is a mixed independent set, then we may independently solve the problem of finding a ρ-realization from L to $I(X)$ and then from $I(X)$ to R. If no solution is found for one of them, then the converse of Lemma 8 implies that no ρ-realizations exists for G. The algorithm of the following section is based on this approach.

4.3 XP algorithm

Algorithm details. We present here a direct algorithm for Bipartite Independent Set Reconfiguration, detailed in Algorithm 1. The main function `Realize` is recursive. Its sub-calls arise either from a split with a mixed MIS I (in which case it is called on a smaller graph but with the same parameter), or from the loop over all possible starting points in the case where no separator is found (lines 13-18), in which case the parameter does reduce. The overall runtime is dominated by this loop, and is analyzed in Proposition 9 below.

Mixed MIS algorithm. The sub-routine allowing to find, if it exists, a maximum independent set intersecting both L and R is based on concepts from *matching theory* [16], namely the *Dulmage-Mendelsohn* decomposition [3, 16], as well as the decomposition of bipartite graphs with a perfect matching into *elementary subgraphs* [16](part 4.1). Its full details are described in the full version of the article.

Algorithm 1 XP algorithm for Bipartite Independent Set Reconfiguration.

Input : bipartite graph G (with sides L and R), integer ρ
Output : a ρ-realization for G, if it exists

Function `Realize`(G, ρ)*:*
  // Terminal cases:
  if $\rho < 0$ **then return** $\perp$
  if $|L \cup R| = \emptyset$ **then return** $\emptyset$
  // Isolated vertices:
  if $\exists \ell \in L$ *s.t* $N(\ell) = \emptyset$) **then return** `Realize`$(G \setminus \{\ell\}, \rho - 1) \cdot l$
  if $\exists r \in R$ *s.t* $N(r) = \emptyset$) **then return** $r \cdot$ `Realize`$(G \setminus \{r\}, \rho - 1)$
  // Trying to find a separator (see full version of the article for details)
  $I =$ `MixedMIS`(G)
  if $I \neq \perp$ **then**
    $S = (L \setminus I) \cup (R \cap I)$ *// intermediate point.*
    return `Realize`$(G[S], \rho) \cdot$ `Realize`$(G[V \setminus S], \rho)$
  else
    // Iterating over all possible start/end point pairs.
    for $(\ell, r) \in L \times R$ **do**
      if `Realize`$(G \setminus \{\ell, r\}, \rho - 1) \neq \perp$ **then**
        return $\ell \cdot$ `Realize`$(G \setminus \{\ell, r\}, \rho - 1) \cdot r$
    return $\perp$

▶ **Proposition 9.** *Algorithm 1 runs in $O(|V|^{2\rho}\sqrt{|V|}|E|)$ time, while using $O(|V|^2)$ space, where ρ is the difference between the minimum allowed and maximum possible independent set size, along the reconfiguration.*

Proof. Let us start with space: throughout the algorithm, one needs only to maintain a description of G and related objects (independent set I, maximum matching M, associated directed graph $H(G, M)$) for which $O(|V|^2)$ is enough.

As for time, let $C(n_1, n_2, \rho)$ be the number of recursive calls of the function *Realize* of Algorithm 1 when initially called with $|L| = n_1$, $|R| = n_2$, and some value of ρ. We will show by induction that $C(n_1, n_2, \rho) \leq (n_1 + n_2)^{2\rho}$. Since each call involves one computation of a maximum matching, this will prove our result.

Given (n_1, n_2, ρ), suppose therefore that $\forall (n_1', n_2', \rho') \neq (n_1, n_2, \rho)$ with $n_1' \leq n_1, n_2' \leq n_2, \rho' \leq \rho$ we have $C(n_1', n_2', \rho') < (n_1' + n_2')^{2\rho'}$

1. If G allows for a mixed maximum independent set, the instance is split into two smaller instances, yielding $C(n_1, n_2, \rho) = C(n_1', n_2, \rho) + C(n_1'', n_2'', \rho)$ with $n_1' + n_1'' = n_1$ and $n_2 = n_2' + n_2''$. And $C(n_1, n_2, \rho) \leq \left((n_1' + n_2')^{2\rho} + (n_1'' + n_2'')^{2\rho}\right) \leq (n_1' + n_1'' + n_2' + n_2'')^{2\rho} \leq (n_1 + n_2)^{2\rho}$.

2. else, we have the following relation: $C(n_1, n_2, \rho) = n_1 n_2 \cdot C(n_1 - 1, n_2 - 1, \rho - 1)$. Which yields:

$$\begin{aligned} C(n_1, n_2, \rho) &= n_1 n_2 \cdot C(n_1 - 1, n_2 - 1, \rho - 1) \\ &\leq n^2 \cdot n^{2(\rho-1)} \quad \text{by induction hypothesis} \\ &\leq n^{2\rho} \end{aligned}$$

◀

The exponential part ($O(n^{2\rho})$) of the worst case complexity of Algorithm 1 is in fact tight, as it is met with a complete bi-clique $K_{n,n}$ with sides of size n. Indeed, in this case, no mixed MIS is found in any of the recursive calls.

5 Benchmarks and Applications

In this section, we report benchmark results for both algorithmic approaches. We first explain some details about the algorithm we implemented for directed pathwidth. Then, we present a general benchmark of our algorithms on random (Erdös-Rényi) bipartite graphs. Last, we give some background related to RNA bioinformatics and the application of our algorithm to the barrier energy problem.

Code availability. The code used for our benchmarks, including a Python/C++ implementation of our two algorithms, is available at `https://gitlab.inria.fr/bmarchan/bisr-dpw`

5.1 Implementation details

Directed pathwidth. We implemented and used an algorithm from Tamaki [20], with a runtime of $O(n^{\rho+2})$. This algorithm was originally published in 2011 [20]. In 2015, H.Tamaki and other authors described this algorithm as "flawed" in [11], and replaced it with another XP algorithm for directed pathwidth, with a run-time of $O(\frac{mn^{2\rho}}{(\rho-1)!})$.

Upon further analysis from our part, and discussions with H. Tamaki and the corresponding author of [11], it appears a small modification allowed to make the algorithm correct. In a nutshell, the algorithm involves *pruning* actions, and these need to be carried out *as soon as they are detected*. In [20], temporary solutions were accumulated before a general pruning step. With this modification, the analysis presented in [20] applies without modification, and yields a time complexity of $O(n^{\rho+2})$. The space complexity is unchanged at $O(n^{\rho+1})$. For completeness, a detailed re-derivation of the results of [20] is included in the full version of the article.

Mixed-MIS algorithm implementation. On Figure 2, the "m-MIS"-curve, corresponds to our mixed-MIS-based algorithm in $O(n^{2\rho}\sqrt{|V||E|})$. Compared to the algorithm presented in Algorithm 1, a more efficient rule is used in the non-separable case: we loop over all possible $r \in R$ and add $N(r) \cdot r$ to the schedule (instead of a single vertex $\ell \in L$).

5.2 Random bipartite graphs

Benchmark details. Figure 2 shows, as a function of the number of vertices, the average execution time of both our algorithms (top panel), as well as the distribution of parameter values (ρ - bottom panel), on a class of random bipartite graphs. These graphs are generated according to an Erdös-Rényi distribution (each pair of vertices has a constant probability p of forming an edge). We use a connection probability of d/n, dependent on the number of vertices. It is such that the average degree of vertices is d. The data of our benchmark (Figure 2) has been generated with $d = 5$.

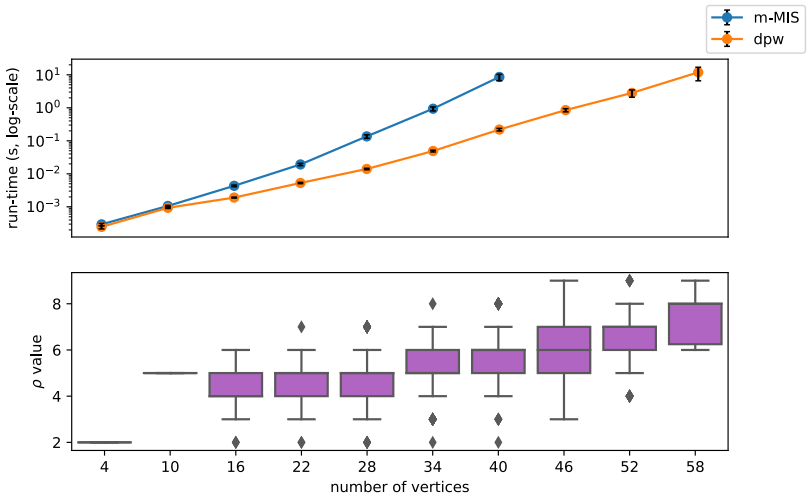


Figure 2 (top panel) Average run-time (seconds, log-scale) of our algorithms on random Erdös-Rényi bipartite graphs, with a probability of connection such that the average degree of a vertex is 5 (i.e $p = 5/n$). (bottom panel) Average parameter value of generated instances, as a function of input size.

Comments on Figure 2. The difference in trend between the execution times of the two algorithms is quite coherent with the difference in their exponents ($n^{\rho+2}$ vs. $n^{2\rho+2.5}$).

5.3 Computing energy barriers in RNA kinetics

In this section, we give more detail about how our algorithms may apply to a bioinformatics problem, the RNA barrier energy problem. We present benchmark results, on a random class of RNA instances, showing the practicality of our approach.

RNA basics. RiboNucleic Acids (RNAs) are biomolecules of outstanding interest for molecular biology, which can be represented as strings over an alphabet $\Sigma := \{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{U}\}$ (in this context, n denotes the length of the string). Importantly, these strings may *fold* on themselves to adopt one or several conformation(s). A conformation is typically described by a set of base pairs $(i, j), i < j$. Then, a standard class of conformations to consider in RNA bioinformatics are *secondary structures*, which are pairwise non-crossing ($\nexists (i, j), (k, l) \in S$ such that $i \leq k \leq j \leq l$, in particular, they involve distinct positions). In this section, we more precisely work on the problem of finding a reconfiguration pathway between two *secondary structures* (i.e conflict-free sets of base pairs). The reconfiguration may only involve secondary structures, and remain of energy as low as possible. We work with a simple energy model consisting of the opposite of *number of base pairs* in a configuration ($-N_{bps}$). The RNA Energy-Barrier problem can then be stated as such:

RNA Energy-Barrier
Input: Secondary structures L and R; Energy barrier $k \in \mathbb{N}^+$
Output: True if there exists a sequence $S_0 \cdots S_\ell$ of secondary structures such that
- $S_0 = L$ and $S_\ell = R$;
- $|S_i| \geq |L| - k, \forall i \in [0, \ell]$;
- $|S_i \triangle S_{i+1}| = 1, \forall i \in [0, \ell - 1]$.

False otherwise.

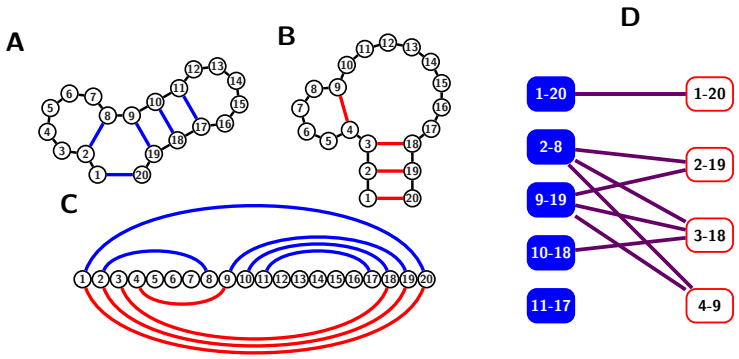


Figure 3 Conflict bipartite graph (D) associated with an instance of the RNA Energy-Barrier problem, consisting of an initial (A) and final (B) structure, both represented as an arc-annotated sequence (C). The sequence of valid secondary structures, achieving minimum energy barrier can be obtained from the solution given in Figure 1.

Bipartite representation. Given two secondary structures L and R, represented as sets of base pairs, we define a *conflict graph* $G(L, R)$ such that: the vertex set of $G(L, R)$ is $L \cup R$; and two vertices $(i, j), (k, l)$ are connected if they are crossing (see Figure 3). Since base pairs in both L and R are both pairwise non-crossing, $G(L, R)$ is bipartite with parts L and R. In this context, a maximum independent set of $G(L, R)$ is a *minimum free-energy structure* of the RNA, and we write $\text{MFE}(L, R) = \alpha(G(L, R))$. We then see how the RNA Energy-Barrier problem is simply Bipartite Independent Set Reconfiguration restricted to a specific class of bipartite graphs: the conflict graphs of secondary structures, with a range of $\rho = k + \text{MFE}(L, R) - |L|$.

Problem motivation. Since the number of secondary structures available to a given RNA grows exponentially with n, RNA energy landscapes are notoriously *rugged*, *i.e.* feature many local minima, and the folding process of an RNA from its *synthesis* to its theoretical final state (a thermodynamic equilibrium around low energy conformations) can be significantly slowed down. Consequently, some RNAs end up being degraded before reaching this final state. This observation motivates the study of RNA kinetics, which encompass all time-dependent aspects of the folding process. In particular, it is known (Arrhenius law) that the energy barrier is the dominant factor influencing the transition rate between two structures, with an exponential dependence.

Related works in bioinformatics. The problem was shown to be NP-hard by Maňuch *et al* [17]. Thachuk *et al* [21] also proposed an XP algorithm in $O(n^{2k+2.5})$ parameterized by the energy barrier k, restricted to instances such that the maximum independent set of $G(L, R)$ has cardinality equal to $|L|$ and $|L| = |R|$.

Benchmark details. Figure 4 shows (top panel) the average execution time of our algorithms on random RNA instances. The bottom panel shows the parameter distribution as a function of the length of the RNA string. Random instances are generated according to the following model: two secondary structures L, R are chosen *uniformly* at random (within the space of all possible secondary structure). Base pairs are constrained to occur between nucleotides separated by a distance of at least $\theta = 5$.

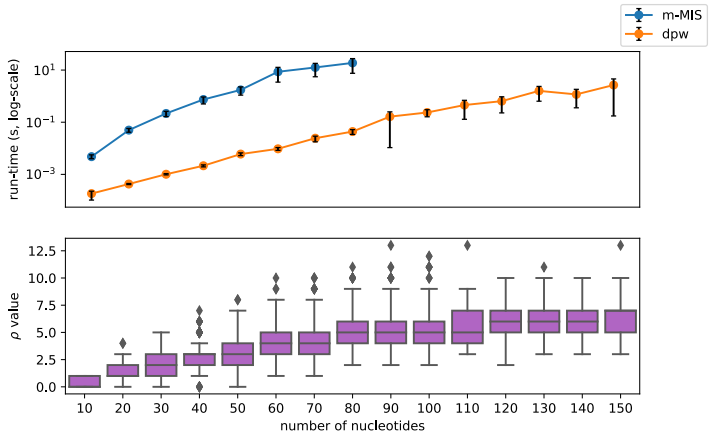


Figure 4 Execution time of our algorithms on random RNA reconfiguration instances (top panel). On the bottom panel, the distribution of the parameter value (ρ) is plotted against the length of the RNA string. Error bars (top panel) are obtained using a bootstrapping method.

5.4 RNA specific optimizations

Dynamic Programming and RNA. Given two secondary structures L and R, a mixed MIS of $G(L, R)$ is a *maximum conflict-free* subset of $L \cup R$, containing at least a base pair from L and R. As is the case for many algorithmic problems involving RNA, the fact that RNAs are *strings* and that base pairs define *intervals* suggests a dynamic programming approach to the mixed maximum independent set problem in RNA conflict graphs. Subproblems will correspond to *intervals* of the RNA string. Let us start with a simple dynamic programming scheme allowing to compute an unconstrained MIS.

Unconstrained MIS DP scheme. A maximum conflict-free subset of $L \cup R$ can be computed by dynamic programming, using the following DP table: for each $1 \leq i \leq j \leq n$, let $MCF_{i,j}$ be the size of a maximum conflict-free subset of all base pairs included in $[i, j]$.

▶ **Lemma 10.** *$MCF_{1,n}$ can be computed in time $O(n^2)$*

Proof. We have the following recurrence formula:

$$MCF_{i,i'} = 0, \forall i' < i$$

$$MCF_{i,j} = \max \begin{cases} MCF_{i+1,j} \\ \max_{(i,k)\in L\cup R} 1 + MCF_{i+1,k-1} + MCF_{k+1,j} \end{cases}$$

Note that the last *max* is over at most two possible pairs (i, k) (1 from L and 1 from R), per the fact that L and R are both conflict-free. ◀

Mixed MIS DP scheme. The following modifications to the DP scheme above allow to compute a mixed MIS of $G(L, R)$ while retaining the same complexity. In addition to the interval, we index the table by Boolean α and β which, when true, further restricts the optimization to subsets with > 0 pair from L (iff $\alpha = \mathsf{True}$) or R (iff $\beta = \mathsf{True}$):

$$MCF^{\alpha,\beta}_{i,i'} = \begin{cases} 0 & \text{if } (\alpha,\beta) = (\mathsf{False},\mathsf{False}) \\ -\infty & \text{otherwise} \end{cases}, \forall i' < i$$

$$MCF^{\alpha,\beta}_{i,j} = \max \begin{cases} MCF^{\alpha,\beta}_{i+1,j} \\ \max\limits_{\substack{(i,k)\in E \\ \alpha',\alpha'',\beta',\beta''\in\mathbb{B}^4}} 1 + MCF^{\alpha',\beta'}_{i+1,k-1} + MCF^{\alpha'',\beta''}_{k+1,j} \end{cases} \quad \begin{array}{l} \text{if } \neg\alpha \vee \alpha' \vee \alpha'' \vee ((i,k) \in L) \\ \text{and } \neg\beta \vee \beta' \vee \beta'' \vee ((i,k) \in R) \end{array}$$

Through a suitable memorization, the system can be used to compute in $\mathcal{O}(n^2)$ the maximum cardinality $MCF^{\mathsf{True},\mathsf{True}}_{1,n}$ of a subset over the whole sequence. A backtracking procedure is then used to rebuild the maximal subset.

6 Conclusion

Our work so far sheds a new light on both Bipartite Independent Set Reconfiguration and Directed Pathwidth problems. The former can thus be solved with a parameterized algorithm, having important applications in RNA kinetics since the range parameter is particularly relevant in this context. We hope the newly drawn connection will help settle the fixed parameter tractability of computing the directed pathwidth. A slightly more accessible open problem would be to design an FPT algorithm for BISR in the context of secondary structure conflict graphs (i.e. those graphs arising in RNA reconfiguration).

References

1 János Barát. Directed path-width and monotonicity in digraph searching. *Graphs and Combinatorics*, 22(2):161–172, 2006.

2 Hans L Bodlaender. Fixed-parameter tractability of treewidth and pathwidth. In *The Multivariate Algorithmic Revolution and Beyond*, pages 196–227. Springer, 2012.

3 Jianer Chen and Iyad A Kanj. Constrained minimum vertex cover in bipartite graphs: complexity and parameterized algorithms. *Journal of Computer and System Sciences*, 67(4):833–847, 2003.

4 David Coudert, Dorian Mazauric, and Nicolas Nisse. Experimental evaluation of a branch-and-bound algorithm for computing pathwidth and directed pathwidth. *Journal of Experimental Algorithmics (JEA)*, 21:1–23, 2016.

5 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5. Springer, 2015.

6 Joshua Erde. Directed path-decompositions. *SIAM Journal on Discrete Mathematics*, 34(1):415–430, 2020.

7 Komei Fukuda and Tomomi Matsui. Finding all the perfect matchings in bipartite graphs. *Applied Mathematics Letters*, 7(1):15–18, 1994.

8 Marinus Gottschau, Felix Happach, Marcus Kaiser, and Clara Waldmann. Budget minimization with precedence constraints. *CoRR*, abs/1905.13740, 2019. `arXiv:1905.13740`.

9 Takehiro Ito, Marcin Kamiński, Hirotaka Ono, Akira Suzuki, Ryuhei Uehara, and Katsuhisa Yamanaka. Parameterized complexity of independent set reconfiguration problems. *Discrete Applied Mathematics*, 283:336–345, 2020.

10 Jeff Kinne, Ján Manuch, Akbar Rafiey, and Arash Rafiey. Ordering with precedence constraints and budget minimization. *CoRR*, abs/1507.04885, 2015. `arXiv:1507.04885`.

11 Kenta Kitsunai, Yasuaki Kobayashi, Keita Komuro, Hisao Tamaki, and Toshihiro Tano. Computing directed pathwidth in $o(1.89^n)$ time. *Algorithmica*, 75(1):138–157, 2016.

12 Yasuaki Kobayashi, Keita Komuro, and Hisao Tamaki. Search space reduction through commitments in pathwidth computation: An experimental study. In *International Symposium on Experimental Algorithms*, pages 388–399. Springer, 2014.

13 Daniel Lokshtanov and Amer E Mouawad. The complexity of independent set reconfiguration on bipartite graphs. *ACM Transactions on Algorithms (TALG)*, 15(1):1–19, 2018.

14 Daniel Lokshtanov, Amer E Mouawad, Fahad Panolan, MS Ramanujan, and Saket Saurabh. Reconfiguration on sparse graphs. *Journal of Computer and System Sciences*, 95:122–131, 2018.

15 Daniel Lokshtanov, Amer E Mouawad, Fahad Panolan, and Sebastian Siebertz. On the parameterized complexity of reconfiguration of connected dominating sets. *arXiv preprint arXiv:1910.00581*, 2019.

16 László Lovász and Michael D Plummer. *Matching theory*, volume 367. American Mathematical Soc., 2009.

17 Ján Maňuch, Chris Thachuk, Ladislav Stacho, and Anne Condon. Np-completeness of the energy barrier problem without pseudoknots and temporary arcs. *Natural Computing*, 10(1):391–405, 2011.

18 Amer E Mouawad, Naomi Nishimura, Venkatesh Raman, Narges Simjour, and Akira Suzuki. On the parameterized complexity of reconfiguration problems. *Algorithmica*, 78(1):274–297, 2017.

19 Hisao Tamaki. A directed path-decomposition approach to exactly identifying attractors of boolean networks. In *2010 10th International Symposium on Communications and Information Technologies*, pages 844–849. IEEE, 2010.

20 Hisao Tamaki. A polynomial time algorithm for bounded directed pathwidth. In Petr Kolman and Jan Kratochvíl, editors, *Graph-Theoretic Concepts in Computer Science*, pages 331–342, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

21 Chris Thachuk, Jan Manuch, Arash Rafiey, Leigh-Anne Mathieson, Ladislav Stacho, and Anne Condon. An Algorithm for the Energy Barrier Problem Without Pseudoknots and Temporary Arcs. In *Biocomputing 2010*, pages 108–119. World Scientific, October 2009. doi:10.1142/9789814295291_0013.

22 Ignacio Tinoco Jr and Carlos Bustamante. How rna folds. *Journal of molecular biology*, 293(2):271–281, 1999.

23 Jan van den Heuvel. The complexity of change. *Surveys in combinatorics*, 409(2013):127–160, 2013.

24 Boting Yang and Yi Cao. Digraph searching, directed vertex separation and directed pathwidth. *Discrete Applied Mathematics*, 156(10):1822–1837, 2008.

25 Zan-Bo Zhang, Xiaoyan Zhang, and Xuelian Wen. Directed hamilton cycles in digraphs and matching alternating hamilton cycles in bipartite graphs. *SIAM J. Discret. Math.*, 27(1):274–289, 2013. doi:10.1137/110837188.

Lower Bounds for Conjunctive and Disjunctive Turing Kernels

Elisabet Burjons
Department of Computer Science, RWTH Aachen, Germany

Peter Rossmanith
Department of Computer Science, RWTH Aachen, Germany

Abstract

The non-existence of polynomial kernels for OR- and AND-compositional problems is now a well-established result. Some of these problems have adaptive or non-adaptive polynomial Turing kernels. Up to now, most known polynomial Turing kernels are non-adaptive and most of them are of the conjunctive or disjunctive kind. For some problems it has been conjectured that the existence of polynomial Turing kernels is unlikely. For instance, either all or none of the WK[1]-complete problems have polynomial Turing kernels. While it has been conjectured that they do not, a proof tying their non-existence to some complexity theoretic assumption is still missing and seems to be beyond the reach of today's standard techniques.

In this paper, we prove that OR-compositional problems and all WK[1]-hard problems do not have conjunctive polynomial kernels, a special type of non-adaptive Turing kernels, under the assumption that coNP $\not\subseteq$ NP/poly. Similarly, it is unlikely that AND-compositional problems have disjunctive polynomial kernels. Moreover, we present a way to prove that the parameterized versions of some $\oplus P$-hard problems, for instance, Odd Path on planar graphs, do not have conjunctive or disjunctive polynomial kernels, unless coNP $\subseteq$ NP/poly. Finally, we identify a problem that is unlikely to have either a conjunctive or disjunctive polynomial kernel, unless coNP $\subseteq$ NP/poly, due to a reduction from an NP-hard problem instead: Weighted Odd Path on planar graphs.

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

Keywords and phrases Parameterized Complexity, Turing kernels

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.12

Funding *Elisabet Burjons*: This work was supported by the Swiss National Science Foundation postdoc mobility grant P2EZP2-191861.

1 Introduction

Kernelization [12] is a central notion in parameterized complexity. Given a parameterized problem, a kernelization procedure reduces an instance of size n and parameter k to a size that only depends on the parameter in polynomial time. A *polynomial kernel* is the result of a kernelization procedure where one can find a single instance equivalent to the original instance but of size polynomial in k.

However, there are problems that do not admit polynomial kernels under common complexity theoretic assumptions. In particular, Fortnow and Santhanam [13] together with Bodlaender et al. [2] proved that, so called, OR-compositional problems do not have polynomal kernels, unless NP $\subseteq$ coNP/poly, which means that the polynomial hierarchy collapses to the third level [25]. Polynomial parameter transformations (PPT) also allowed to further extend the class of problems unlikely to have polynomial kernels [4]. Some of these problems are, for instance, Longest Path, Clique(VC), Set Cover($|U|$), and Connected Vertex Cover. If nothing is otherwise indicated, these problems are parameterized by the targeted

solution size k. Here (VC) indicates that the problem is parameterized by the size of the vertex cover, and $|U|$ indicates that the problem is parameterized by the universe size. The exact definitions of all problems in this paper can be found in [12], unless otherwise stated.

Later, Drucker [11] extended these results by proving that AND-compositional problems do not have polynomial kernels, unless there are statistical zero knowledge proofs for all the problems in NP, which is a stronger condition than NP $\subseteq$ coNP/poly. Thus, the list of problems that do not have polynomial kernels also includes Treewidth, Independent Set(ω), and 3-coloring(ω), where ω indicates parameterization by size of the treewidth. Moreover, Drucker extended the previous results to include the unlikely existence of probabilistic polynomial kernels, for both OR and AND-compositional problems, with a small enough error probability.

Guo was the first to ask the question whether one could extend the notion of efficient kernelization to include polynomial Turing kernelizations [5], where instead of building one reduction, an algorithm builds many reductions from the original problem into instances of smaller size (only depending polynomially on the parameter), then the algorithm has access to an oracle solving such instances, and outputs an answer for the original instance in polynomial time. The first example of such a kernel, by Raible et al. [1], is the Leaf Out-Branching problem, in which, given a directed graph G and an integer k, one needs to find whether G contains a directed tree with at least k leaves. If the root of the tree is fixed, this problem admits a kernel of size $O(k^3)$, thus, by fixing all possible n vertices one can build the kernel for each possible root and use the oracle to find out whether at least one of these roots extends to a tree with at least k vertices, if none of them do, the graph does not contain such a tree, and otherwise the answer is yes. We call such a Turing kernel formed as the disjunction of n (or more) different instances a disjunctive Turing kernel, or a disjunctive kernel for short. Analogously one can find in the literature other examples of disjunctive kernels for OR-compositional problems such as Clique with bounded degree, Clique(VC), etc. One can analogously find conjunctive Turing kernelizations, or conjunctive kernelizations, for AND-compositional problems in some cases, however, not all such problems seem to admit one of these kernelizations. The first instance of an adaptive Turing kernelization that is neither conjunctive nor disjunctive is for Longest Path in planar graphs by Jansen [15]. In this paper, unless otherwise specified, all mentions of conjunctive, disjunctive, or Turing kernelizations refer to polynomial conjunctive, disjunctive, or Turing kernelizations.

Hermelin et al. [14] propose a completeness theory for Turing kernelization. Using the class of problems WK[1], characterized among others by Set Cover($|U|$), the authors conjecture that problems that are hard for this class do not admit Turing kernelizations, however, the authors do not provide complexity-theoretic consequences for the non-existence of Turing kernels for those problems.

On the other hand, Witteveen, Bottesch, and Torenvliet [24] show that the kernelization hierachy is strict by using diagonalization arguments to show that there exist problems that have Turing kernels but do not have non-adaptive Turing kernels such as disjunctive or conjunctive kernels, and problems that have non-adaptive Turing kernels but do not admit polynomial kernels. However, the presented problems to prove the strictness of the hierarchy are not natural.

It is worth mentioning that there are problems for which polynomial Turing kernels are unlikely due to lower bounds on their FPT running time. Any problem with a polynomial Turing kernel, whose non-parameterized version is in EXPTIME, will have a single exponential FPT algorithm consisting of running the algorithm for the polynomial Turing kernel and solving any calls to the oracle with the non-parameterized algorithm. For instance,

Cygan, Pilipczuk and Pilipczuk [7] proved that the Edge Clique Cover problem has no $2^{2^{o(k)}}\text{poly}(n)$ algorithm unless the ETH hypothesis fails. Thus, it is equally unlikely that there exist subexponential Turing kernels for Edge Clique Cover.

In this paper, we extend the proofs of nonexistence of polynomial kernels to the nonexistence of conjunctive kernels in the case of OR-compositional problems and disjunctive kernels in the case of AND-compositional problems under the same complexity theoretic assumptions also in the probabilistic case. We prove, thus, that WK[1]-hard problems do not admit conjunctive kernels, unless NP $\subseteq$ coNP/poly. Using these results, we consider XOR-compositional problems, which naturally correspond to problems counting parity, and prove that if a reduction from an NP-hard problem is possible these types of problems do not have conjunctive or disjunctive kernels, unless coNP $\subseteq$ NP/poly. We also prove that if a reduction from a $\oplus$P-hard problem is possible XOR-cross-compositional problems do not have a conjunctive or disjunctive kernel, unless NP $\subseteq$ coNP/poly. By way of an example, we find that Odd Path on planar graphs, the problem of finding whether a planar graph contains an odd number of paths of length k, is unlikely to have conjunctive or disjunctive kernels.

Furthermore, we find an example of problem that does not have conjunctive or disjunctive kernels using the traditional complexity-theoretic approach to lower bounds, i.e., through a reduction from an NP-hard problem, in this case, Longest Path. The problem Weighted Odd Path on planar graphs, a weighted version of the Odd Path problem, does not have conjunctive or disjunctive kernels, unless NP $\subseteq$ coNP/poly. We hope that this approach, using the classic complexity-theoretic results to achieve the lower bound, can be used to prove similar lower bounds on the existence of conjunctive and disjunctive kernels for more classical FPT problems in the future.

These lower bounds do not provide lower bounds for the existence of Turing kernels in general, but they do provide lower bounds for some of the most common kinds of Turing kernelizations.

2 Lower Bounds for Conjunctive Kernels

First we want to prove that there are no conjunctive kernels for OR-compositional parameterized problems. In order to do that, we first formally define the notion of conjunctive kernelization.

▶ **Definition 1** (Conjunctive Compression)**.** Given two parameterized problems L and R, a conjunctive compression is an algorithm that given an instance (x, k) for L of size n, it outputs in polynomial time a set of instances (x'_i, k'_i) for R with $1 \leq i \leq p(n)$ and $|x'_i|, k'_i \leq q(k)$ for every i for some polynomials $p(\cdot)$ and $q(\cdot)$, such that $(x, k) \in L$ if and only if $(x'_i, k'_i) \in R$ for every i.

2.1 Conjunctive OR-distillations

To prove that this type of kernels do not exist we have to go to the original proof of nonexistence of polynomial compressions for OR-distillable problems. Given a problem L, we define the problem OR(L), where the input is a set of instances $x_1, \ldots, x_t$ of length at most n, and the task is to decide whether there exists an i such that $x_i \in L$.

First we define distillation algorithm.

▶ **Definition 2** (OR-Distillation Algorithm [2]). Let $L, R \subseteq \{0,1\}^*$ be a pair of languages and let $t\colon \mathbb{N} \to \mathbb{N} \setminus \{0\}$ be a function. Then a t-bounded OR-distillation algorithm from L into R is an algorithm A that for every n, given $t(n)$ input strings $x_1, \ldots, x_{t(n)}$, with $|x_i| = n$ for all i, A runs in polynomial time and outputs a string y of length at most $t(n) \cdot \log n$ such that $y \in R$ if and only if $x_i \in L$ for some $i \in \{1, \ldots, t(n)\}$.

Essentially, an OR-distillation for a problem L is a reduction from $\mathrm{OR}(L)$ into R. Recall that, an NP-hard problem cannot have an OR-distillation, unless NP $\subseteq$ coNP/poly [13]. A Conjunctive OR-Distillation Algorithm is an extension of this definition.

▶ **Definition 3** (Conjunctive OR-Distillation Algorithm). Let $L, R \subseteq \{0,1\}^*$ be a pair of languages and let $t\colon \mathbb{N} \to \mathbb{N} \setminus \{0\}$ be a function. Then a t-bounded conjunctive OR-distillation algorithm from L into R is an algorithm A that for every n, given $t(n)$ input strings $x_1, \ldots, x_{t(n)}$, with $|x_i| = n$ for all i, A runs in polynomial time and outputs $p(n)$ strings $y_1, \ldots, y_{p(n)}$ of length at most $t(n) \cdot \log n$ such that $y_i \in R$ for every $i \in \{1, \ldots, p(n)\}$ if and only if $x_j \in L$ for some $j \in \{1, \ldots, t(n)\}$.

We can use the same procedure as in [13] to rule out the existence of Conjunctive OR-Distillation Algorithms for NP-hard languages.

We take the pigeonhole lemma from Fomin et al. [12].

▶ **Lemma 4.** *Let X, Y be finite sets, t be a natural number and $\beta\colon X^t \to Y$ be a mapping. We say that $y \in Y$ covers $x \in X$ if there exist $x_1, \ldots, x_t$ such that $x_i = x$ for some i, $\beta((x_1, \ldots, x_t)) = y$. Then at least one element from Y covers at least $|X|/\sqrt[t]{|Y|}$ elements of X.*

Now we are ready to prove the following theorem.

▶ **Theorem 5.** *If there is a t-bounded Conjunctive OR-distillation algorithm from a language $L \subseteq \{0,1\}^*$ into a language $R \subseteq \{0,1\}^*$ for some polynomially bounded function t, then $\bar{L} \in$ NP/poly. In particular if L is* NP*-hard, then* coNP $\subseteq$ NP/poly.

Proof. Let $n \in \mathbb{N}$ and $\bar{L}_n = \{\, x \in \bar{L} \mid |x| = n \,\}$. Let us assume that there is a t-bounded Conjunctive OR-distillation algorithm A from L into R. Then, for each input $(x_1, \ldots, x_{t(n)}) \in \bar{L}_n^{t(n)}$, A outputs $y_1, \ldots, y_{p(n)}$ such that $|y_i| \leq t(n) \cdot \log n$ and $y_i \in \bar{R}$ for at least one i. Define $\beta\colon \bar{L}_n^{t(n)} \to \bar{R}_{\leq t(n)\log(n)}$ as follows: If $A(x_1, \ldots, x_{t(n)}) = y_1, \ldots, y_{p(n)}$ then define $\beta(x_1, \ldots, x_{t(n)}) = y_i$ where i is the smallest index for which $y_i \in \bar{R}$. Now apply Lemma 4 to this function β with $t = t(n)$, $X = \bar{L}$ and $Y = \bar{R}$. The lemma then states that there is a $z_1 \in \bar{R}$ that covers at least $|\bar{L}_n|/|\bar{R}_{t(n)\log(n)}|^{1/t(n)} \geq |\bar{L}_n|/n$ elements from $\bar{L}$.

Let Z_1 the set of $x \in \bar{L}_n$ covered by z_1. Then we consider the set $\bar{L}_n \setminus Z_1$ and find in the same way a z_2 that covers as many instances from $\bar{L}_n \setminus Z_1$ as possible. Repeating this for $2n$ rounds we get a set of z_i's that together cover all of $\bar{L}_n$ just as in the proof by Fortnow and Santhanam [13]: In each round the number of uncovered elements is reduced by a factor of $1 - 1/n$ and $(1 - 1/n)^{2n} < 2^{-n}$. As $|\bar{L}| \leq 2^n$ the number of uncovered elements in the end is less than one and hence it is exactly zero. In conclusion we have shown that there is a polynomial size subset S_n of elements of $\bar{R}$ that covers the whole $\bar{L}_n$.

We want to show next how a nondeterministic TM can decide $\bar{L}$ in polynomial time with the help of polynomial advice. For an input x of size n the TM does the following: First, it guesses $x_1, \ldots, x_{t(n)}$ such that $x = x_i$ for some i and $x_j \in \bar{L}$ for every $j \neq i$. Then it computes $A(x_1, \ldots, x_{t(n)}) = y_1, \ldots, y_{p(n)}$ and checks if $y_i \in S_n$ for at least one i. The set S_n is read from the advice tape. If there is indeed such a $y_i \in S_n$ the machine accepts x and rejects it otherwise.

If $x \in \bar{L}$ then the TM will accept: As x is covered by some $y \in S_n$ there must be such an $x_1, \ldots, x_{t(n)}$ that lets the machine accept x and it can guess it nondeterministically. On the other hand, if $x \in L$ it is easy to see that the TM cannot accept for any guessed $x_1, \ldots, x_{t(n)}$.

If L is NP-hard then $\bar{L}$ is coNP-hard. This means that every problem in coNP can be reduced to $\bar{L}$. Thus, coNP $\subseteq$ NP/poly. ◀

2.2 Conjunctive Kernels

In order to use Theorem 5 to prove that some parameterized problems are unlikely to have conjunctive kernelizations, we use OR-cross-compositions and polynomial parameter transformations.

▶ **Definition 6** (Bodlaender, Jansen, and Kratsch [3])**.** An equivalence relation R on the set Σ^* is a *polynomial equivalence relation* if there exists an algorithm that given two strings $x, y \in \Sigma^*$ resolves whether x is equivalent to y in polynomial time in $|x| + |y|$, and for any finite set $S \subseteq \Sigma^*$, R partitions the elements of S into at most $(\max_{x \in S} |x|)^{O(1)}$ classes.

▶ **Definition 7** (Bodlaender, Jansen, and Kratsch [3])**.** Let $L \subseteq \Sigma^*$ be a language and $Q \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized language. We say that L *OR-cross-composes* into Q if there exists a polynomial equivalence relation R and an algoritm A called an OR-cross-composition such that A takes as input $x_1, \ldots, x_t \in \Sigma^*$ equivalent with respect to R, and outputs one instance $(y, k) \in \Sigma^* \times N$ such that k is polynomial in the size of x_i and $\log t$, and $(y, k) \in Q$ if and only if there exists one index i such that $x_i \in L$.

In particular, for any problem L, we have a trivial OR-cross-composition to OR(L) parameterized by the size of the largest instance $|x_i| = n$.

▶ **Theorem 8.** *Let $L \subseteq \Sigma^*$ be an NP-hard language. If L OR-cross-composes into a parameterized problem Q and Q has a conjunctive kernelization, then* coNP $\subseteq$ NP/poly*.*

Proof Sketch. The proof is equivalent to that in Bodlaender et al. [2], for polynomial compressions. Given an NP-hard language L and an OR-cross-composition from L into a parameterized problem Q, if Q had a conjunctive kernelization, one would be able to build a conjunctive distillation for L. ◀

From here we are only left to list problems that are unlikely to have a conjunctive kernelization because an NP-hard problem OR-cross-composes into them. These problems, called *OR-compositional*, form an extensive list, a subset of them would be.

▶ **Corollary 9.** *Unless* coNP $\subseteq$ NP/poly*, none of the following FPT problems have conjunctive kernelizations:* Longest Path*,* Longest Cycle*,* Exact Cycle*,* Short Cheap Tour*,* Graph Minor Order Test*,* Bounded Treewidth Subgraph Test*,* Steiner Tree*, and* Clique(VC)*.*

Proof. Exact Cycle, Short Cheap Tour, Graph Minor Order Test and Bounded Treewidth Subgraph Test are defined in Bodlaender et al. [2], the authors provide OR-cross-compositions from their own unparameterized versions for these problems together with Longest Path and Longest Cycle. Dom, Lokshtanov, and Saurabh provide an OR-cross-composition for Steiner Tree [8], and finally, the OR-cross-composition for Clique(VC) is due to Bodlaender Jansen and Kratsch [3]. ◀

Most of the problems on this list cross-compose from their own unparameterized versions, or from a similar problem. But that is not the only option. Once one has a list of problems unlikely to have conjunctive kernelizations one can extend it using Polynomial parameter transformations (PPT).

▶ **Definition 10** (Bodlaender, Thomassé, and Yeo [4]). Given two parameterized problems P and Q an algorithm A is a PPT from P to Q if given an instance (x, k) for P, A transforms it in polynomial time into an instance (x', k') for Q such that k' is polynomial in k and $(x, k) \in P$ if and only if $(x', k') \in Q$.

At this point it is easy to see that, if a problem Q has a conjunctive kernelization, and there is a PPT from P to Q then P also has a conjunctive kernelization (or compression).

Due to the existence of appropriate PPTs the following problems are also unlikely to have conjunctive kernelizations:

▶ **Corollary 11.** *Unless* coNP $\subseteq$ NP/poly*, none of the following FPT problems have conjunctive kernelizations:* Path Packing*,* Cycle Packing*,* Red Blue Dominating Set*,* Set Cover$(|U|)$*,* Connected Vertex Cover*, and* Capacitaded Vertex Cover*.*

Proof. There are PPTs for Path Packing and Cycle Packing from Longest Path and Longest Cycle respectively by Fomin et al. [12]. Red Blue Dominating Set has a PPT from its own colored version due to Dom Lokstanov and Saurabh [8], furthermore Set Cover$(|U|)$, Connected Vertex Cover and Capacitaded Vertex Cover have PPTs to Red Blue Dominating Set, as was also shown in [8, 12]. ◀

The list of problems unlikely to have conjunctive kernelization through PPTs is longer, we only presented some examples.

3 Lower Bounds for Disjunctive Kernels

From the results we have seen so far for OR-compositional problems, it would seem reasonable to find similar results for AND-compositional problems, however, the proof that AND-distillable problems are unlikely to have compressions was not as straightforward. Drucker [11], proved that it is also unlikely that OR-distillable problems have probabilistic compressions and more importantly, that AND-distillable problems are also unlikely to have deterministic or probabilistic compressions.

3.1 Disjunctive AND-distillations

A simplified version of Drucker's approach [11] was presented in a conference version of the paper [10] and in the following book [9]. This simplified version does not have the same scope as the full proof but it can be used to prove that AND-distillable problems are unlikely to have deterministic compressions.

We now extend this result to state that AND-distillable problems are unlikely to have deterministic disjunctive compressions.

The notion of a disjunctive algorithm can be defined analogously the to the conjunctive version.

▶ **Definition 12** (Disjunctive Distillation Algorithm). Let $L, R \subseteq \{0,1\}^*$ be a pair of languages and let $t\colon \mathbb{N} \to \mathbb{N}\backslash\{0\}$ be a function. Then a t-bounded disjunctive AND-distillation algorithm from L into R is an algorithm A that for every n, given $t(n)$ input strings $x_1, \ldots, x_{t(n)}$, with

$|x_i| = n$ for all i, A runs in polynomial time and outputs $p(n)$ strings $y_1, \ldots, y_{p(n)}$, where each y_i has length at most $t(n) \log n$, such that $y_i \in R$ for some $i \in \{1, \ldots, p(n)\}$ if and only if $x_j \in L$ for every $j \in \{1, \ldots, t(n)\}$.

Also, given a problem L, we define the problem AND(L), where the input is a set of instances $x_1, \ldots, x_t$ of length at most n, and the task is to decide whether $x_i \in L$ for every $i \in \{1, \ldots, t\}$.

Equivalently, we can consider the algorithm A as a set of $p(n)$ polynomially computable functions $f_i \colon \{0,1\}^{t(n)\times n} \to \{0,1\}^{t(n)\log n}$ in such a way that the required property is satisfied.

We start by introducing some basic notions and lemmas.

Given a statistical distribution $\mathcal{D}$, the *support* of $\mathcal{D}$, $\sup(\mathcal{D})$ is the set of values that $\mathcal{D}$ assumes with nonzero probability, and $\mathcal{D}(u) = \text{Prob}[\mathcal{D} = \text{u}]$. We can assume from now on that we talk about distributions with finite supports.

▶ **Definition 13** (Statistical Distance). The *statistical distance* of distributions $\mathcal{D}$ and $\mathcal{D}'$ is

$$\|\mathcal{D} - \mathcal{D}'\|_{\text{stat}} = \frac{1}{2} \sum_{u \in \sup(\mathcal{D}) \cup \sup(\mathcal{D}')} |\mathcal{D}(u) - \mathcal{D}'(u)|$$

Just to get some intuition, if two distributions have completely different supports, their distance is 1. If two distributions share some support their distance is smaller.

Now, given a function $f \colon \{0,1\}^{t\times n} \to \{0,1\}^{t'}$, given a subset $A \subseteq \{0,1\}^n$ we can define the distribution $\mathbf{F}_A = f(\mathcal{U}_A^{\otimes t})$, where $\mathcal{U}_A^{\otimes t}$ is the uniform distribution over t-tuples of elements of A. Moreover, for any $a \in \{0,1\}^n$ we define the distribution

$$\mathbf{F}_A[a] = f(\mathcal{U}_A^{\otimes(j-1)}, a, \mathcal{U}_A^{\otimes(t-j)}) \ ,$$

where j is sampled from the uniform distribution over the integers from 1 to t, $\mathcal{U}_{[t]}$.

Finally, we define the *standout factor* of a with respect to A as

$$\beta(a, A) = \|\mathbf{F}_A - \mathbf{F}_A[a]\|_{\text{stat}} \ .$$

The Disguising-Distribution lemma states

▶ **Lemma 14** (Drucker [10]). *Let $f \colon \{0,1\}^{t\times n} \to \{0,1\}^{t'}$ be any (possibly randomized) function for $t, t' \in \mathbb{N}^+$. Let $S \subseteq \{0,1\}^n$, and fix $d > 0$. Given any $\varepsilon > 0$, let $s = \lceil (0.5 \ln 2) n/\varepsilon^2 \rceil$. Let*

$$\hat{\delta} = \min\left\{ \sqrt{(\ln 2)(t'+1)/2t}, 1 - 2^{-3-t'/t} \right\} .$$

Then there exists a collection $K_1, \ldots, K_s$ of multisets of size d contained in S, such that for every $y \in S$, we have

$$\mathbb{E}_{i \sim \mathcal{U}_{[s]}}[\beta(y, K_i \setminus y)] \leq \hat{\delta} + 2t/(d+1) + \varepsilon \ .$$

Intuitively, given a language L, this lemma allows us to pick multisets of L of size d (potentially polynomial), and given any other element of L, the expected statistical distance to one of these multisets will be small, allowing us to distinguish potentially, if a particular element is in L or not. In order for these statistical distances to give meaningful complexity theoretic consequences, we turn to promise problems, in particular circuits.

For a boolean circuit $\mathcal{C}$ let $\mathcal{D}_C$ be the output distribution for a random input. Given two parameters $0 \leq d \leq D \leq 1$, we define the promise problem $\text{SD}^{\geq \text{D}}_{\leq \text{d}}$ with the yes instances being $\Pi_Y = \{\langle C, C' \rangle \colon \|\mathcal{D}_C - \mathcal{D}_{C'}\|_{\text{stat}} \geq D\}$, and the subset of no instances $\Pi_N = \{\langle C, C' \rangle \colon \|\mathcal{D}_C - \mathcal{D}_{C'}\|_{\text{stat}} \leq d\}$.

We will use the class of promise problems having statistical zero-knowledge proofs (pr-SZK). In particular

▶ **Theorem 15** (Sahai and Vadhan [20]). *Let $0 \leq d = d(n) \leq D = D(n) \leq 1$ be (not necessarily computable) parameters.*

1. *If $D > d + 1/\mathrm{poly(n)}$, then $\mathrm{SD}^{\geq \mathrm{D}}_{\leq \mathrm{d}} \in$ pr-NP/poly.*
2. *If we have the stronger gap $D^2 > d + 1/\mathrm{poly(n)}$, then $\mathrm{SD}^{\geq \mathrm{D}}_{\leq \mathrm{d}}$ is many-to-one reducible to $\mathrm{SD}^{\geq 2/3}_{\leq 1/3} \in$ pr-SZK.*

From Lemma 14 and Theorem 15, Drucker [10] proves that (probabilistic) AND-compositional and OR-compositional problems are unlikely to have polynomial kernels. We adapt the proof to our case and state the following theorem for disjunctive kernels for deterministic AND-compositional problems. In Subsection 4.1 we show how one can analogously adapt the proof to work in the probabilistic case, this allows us to present first a self-contained proof that focuses on the difference between AND-distillations and disjunctive AND-distillations without worrying about the probabilistic aspect. This result is already sufficient to prove that AND-compositional problems are unlikely to have disjunctive kernels.

▶ **Theorem 16.** *Let L be any language. Suppose that $t(n)\colon \mathbb{N}^+ \to \mathbb{N}^+$ is a function. Suppose that there exists a t-bounded disjunctive AND-distillation defined by $p(n)$ functions $f_i\colon \{0,1\}^{t(n)\times n} \to \{0,1\}^{t(n)\log n}$ for L with parameter $t(n)$, and some target language R. Then, $L \in$ coNP/poly.*

Proof. We prove this theorem by reducing $\overline{L}$ to $\mathrm{SD}^{\geq \mathrm{D}}_{\leq \mathrm{d}}$. First of all, let us define $L_n = \{0,1\}^n \cap L$, and let us assume that $L_n \subsetneq \{0,1\}^n$. For every function f_i we apply the disguising distribution lemma to L_n. This means that for every i, we define $S = L_n$, $f = f_i$, $t = t(n)$, $t' = t(n)\log n$, $\varepsilon = 1/4n^c$, $d = 8t(n)n^c$ and $\hat{\delta} = \min\left\{\sqrt{(\ln 2)(t\log n + 1)/2t}, 1 - 2^{-3-\log n}\right\}$. Lemma 14 states that for every f_i there exists a collection $K^i_1, \ldots, K^i_s$ of multisets of size d contained in L_n such that for every $y \in L_n$ we have

$$\mathbb{E}_{j\sim\mathcal{U}_{[s]}}[\beta(y, K^i_j \setminus y)] \leq \hat{\delta} + 2t/(d+1) + \varepsilon\,.$$

Observe, that in this case as long as $t(n)$ is polynomial in n, both s and d are polynomial in n, too. Thus, as long as $p(n)$ is also polynomial in n, which by construction it is if this condition is satisfied for $t(n)$, the collection of all subsets for every i is polynomial in size, too. This will be our advice.

On input $y \in \{0,1\}^n$, our reduction outputs for every mapping f_i, $\langle C_i, C'_i\rangle$ of the following randomized circuits. Circuit C_i samples $j \sim \mathcal{U}_{[s]}$, then samples $\hat{x}_i \sim \mathcal{U}^{\otimes t(n)}_{K^i_j}$, and outputs $z_i = f_i(\hat{x}_i)$. Circuit C'_i samples $j \sim \mathcal{U}_{[s]}$, and $k \sim \mathcal{U}_{t(n)}$, then samples $\hat{x}_i \sim \left(\mathcal{U}^{\otimes(k-1)}_{K^i_j}, y, \mathcal{U}^{\otimes(t(n)-k)}_{K^i_j}\right)$ and outputs $z_i = f_i(\hat{x}_i)$.

The idea is that we have a control circuit C that only samples elements from L, the other circuit C' takes also the original input, if both outputs are in the target language this should be reflected in their statistical distance being small. On the other hand, if the input is not in L, then the output generated by circuit C' is not in the target language, thus, the support of C' is completely disjunct from the support of the distribution of control circuit C and their statistical distance is 1. This however, has to be tempered by the disjunctive property of the reduction.

▷ Claim 17. The following holds:

1. If $y \notin L$, then for every i, we have $\|\mathcal{D}_{C_i} - \mathcal{D}_{C'_i}\|_{\mathrm{stat}} = D(n) = 1$;
2. If $y \in L$ then there exists an i such that, $\|\mathcal{D}_{C_i} - \mathcal{D}_{C'_i}\|_{\mathrm{stat}} \leq d(n) = \hat{\delta} + 1/2n^c$

The first part follows from the AND-respecting and disjunctive properties of the reduction. If there is an element that is not in L, there cannot be any mapping that outputs an element of the target language. Thus, every possible output for C_i' will not be in R, while every output from C_i is, thus, these two distributions have completely different supports and they do not intersect, thus their statistical distance must be 1. The second part follows from Lemma 14 and the disjunctive properties of the reduction, the formal proof will follow.

Now, to prove Theorem 16, if $1 - \hat{\delta} \geq 1/n^c$ for sufficiently large n, then $1 - d(n) = D(n) - d(n) \leq 1/n^c$, thus the gap is inversely polynomial in n and also inversely polynomial in the length of the output pairs $\langle C_i, C_i' \rangle$. Thus, given an instance y of the decision problem L, there exists at least one i, for which the pair $\langle C_i, C_i' \rangle$ is a valid instance of the promise problem $\mathrm{SD}^{\geq 1}_{\leq \mathrm{d(n)}}$. We know by Theorem 15 that for the given values of $d(n)$ and $D(n)$, $\mathrm{SD}^{\geq \mathrm{D(n)}}_{\leq \mathrm{d(n)}} \in$ pr-NP/poly. Thus, assume we are given an algorithm A that, with advice a_n, solves $\mathrm{SD}^{\geq \mathrm{D(n)}}_{\leq \mathrm{d(n)}}$ in nondeterministic polynomial time. Given an instance y, we can run the reduction and then apply this algorithm to every $\langle C_i, C_i' \rangle$ given by instance y and correctly decide if $y \notin L$. In particular, if $y \notin L$, for every i, $\langle C_i, C_i' \rangle \in \Pi_Y$, otherwise $y \in L$ and there exists one i such that $\langle C_i, C_i' \rangle \in \Pi_N$. Note here, that there might be pairs that are not a valid instance for the promise problem, but this is not a concern, as there is at least one which will be solved by A together with the advice. Observe that this procedure runs in polynomial time, as one can run A in parallel for every pair $\langle C_i, C_i' \rangle$, and the amount of pairs is $t(n)$ which we have assumed to be polynomial in n. Thus, $\overline{L} \in$ NP/poly, and $L \in$ coNP/poly. ◀

Proof of Claim 17. The first part of the claim is already proven after the claim statement. For the second part, given a disjunctive AND-distillation defined by $p(n)$ functions $f_i \colon \{0,1\}^{t(n)\times n} \to \{0,1\}^{t(n)\log n}$ for L with parameter $t(n)$, and some target language R, and $y \in L$ then, given $(x_1, \ldots, x_t)$ such that every $x_m \in L$ for every $m \in [t(n)]$ and $x_j = y$ for some $j \in [t(n)]$, there exists an $i \in [p(n)]$ such that $f_i(x_1, \ldots x_t) \in R$. Thus, by Lemma 14 if we take the same values of d, $\hat{\delta}$ and ε as in the main theorem, there exists a collection $K_1^i, \ldots, K_s^i$ of multisets of size d contained in L_n such that for every $y \in L_n$ we have

$$\mathbb{E}_{j \sim \mathcal{U}_{[s]}}[\beta(y, K_j^i \setminus y)] \leq \hat{\delta} + 2t/(d+1) + \varepsilon \,.$$

But, recall that $\beta(y, K_j^i \setminus y)$ is nothing else than the statistical distance between the two distributions, which means that the expected value for all possible values of j is the same as $\|\mathcal{D}_{C_i} - \mathcal{D}_{C_i'}\|_{\mathrm{stat}}$ by construction, and also $\hat{\delta} + 2t/(d+1) + \varepsilon = \hat{\delta} + 1/2n^c = d(n)$. ◁

3.2 Disjunctive Kernels

In order to use this result to prove that some parameterized problems do not have conjunctive kernelizations, we use AND-cross-compositions and polynomial parameter transformations just as we did in Subsection 2.2.

We are going to define AND-cross-compositions analogously to OR-cross-compositions using equivalence relations.

▶ **Definition 18.** Let $L \subseteq \Sigma^*$ be a language and $Q \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized language. We say that L AND-cross-composes into Q if there exists a polynomial equivalence relation R and an algoritm A called a cross-composition such that A takes as input $x_1, \ldots, x_t \in \Sigma^*$ equivalent with respect to R, and outputs one instance $(y, k) \in \Sigma^* \times N$ such that k is polynomial in the size of x_i and $\log t$, and $(y, k) \in Q$ if and only if for every index i, $x_i \in L$.

Now we are ready to state the following theorem.

▶ **Theorem 19.** *Let $L \subseteq \Sigma^*$ be an NP-hard language. If L AND-cross-composes into a parameterized problem Q and Q has a disjunctive kernelization, then* coNP $\subseteq$ NP/poly.

Proof Sketch. The proof is analogous to the one for Theorem 8. The main idea is that if we had an AND-cross-composition into such a parameterized problem, and we had a disjunctive kernelization for that problem, we would be able to build a disjunctive distillation for our original problem, which by Theorem 16 would imply that coNP $\subseteq$ NP/poly. ◀

From here we are only left to list problems that are unlikely to have a disjunctive kernelization because an NP-hard problem cross-composes into them. These problems, sometimes called AND-compositional, form an extensive list, a subset of them would be:

▶ **Corollary 20.** *Unless* coNP $\subseteq$ NP/poly*, none of the following FPT problems have disjunctive kernelizations:* Treewidth, Pathwidth, Cutwidth, *and* Modified Cutwidth.

Proof. For the definitions of these problems together with their the AND-compositions we refer to Bodlaender et al. [2]. ◀

Just as in the conjunctive case, the list of problems unlikely to have disjunctive kernelizations be extended using polynomial parameter transformations (PPT).

Other problems that admit AND-compositions or admit PPT to AND-compositional problems are some problems parameterized by treewidth (ω).

▶ **Corollary 21.** *Unless* coNP $\subseteq$ NP/poly*, none of the following FPT problems have disjunctive kernelizations:* Independent Set(ω) *and* 3-coloring(ω).

Proof. Bodlaender et al. present AND-compositions for these two problems in [2]. ◀

4 Problems without Conjunctive or Disjunctive Kernels

In the previous two sections we have proven that there are problems that are unlikely to have conjunctive kernels and disjunctive kernels, respectively. However, none of these problems intersect. In this section, we prove that if a $\oplus$P-hard problem is in coNP/poly then coNP $\subseteq$ NP/poly. Furthermore, we use this result to find a class of problems without conjunctive or disjunctive kernels if they reduce from a problem that is NP-hard or $\oplus$P-hard. For each class, we present an FPT problem unlikely to have conjunctive or disjunctive kernels. The problem Odd Path on planar graphs does not have conjunctive or disjunctive kernels, unless coNP $\subseteq$ NP/poly, due to a reduction from its own unparameterized version and its weighted version, the Weighted Odd Path problem on planar graphs, does not have conjunctive or disjunctive kernels, unless coNP $\subseteq$ NP/poly, due to a reduction from Longest Path, an NP-hard problem.

4.1 Probabilistic Compressions

First we provide the framework for Probabilistic compressions.

▶ **Definition 22** (Probabilistic conjunctive OR-distillation)**.** Let $L, R \subseteq \{0,1\}^*$ be a pair of languages and let $t\colon \mathbb{N} \to \mathbb{N} \setminus \{0\}$ be a function. Then a probabilistic t-bounded conjunctive OR-distillation algorithm from L into R with bounded error $0 \leq \xi < 0.5$ is an algorithm A that for every n, given $t(n)$ input strings $x_1, \ldots, x_{t(n)}$, with $|x_i| = n$ for all i, A runs in polynomial time and outputs $t(n)$ strings $y_1, \ldots, y_{t(n)}$ of length at most $t(n) \cdot \log n$ such that with a probability greater than $1 - \xi$, $y_i \in R$ for every $i \in \{1, \ldots, t(n)\}$ if and only if $x_j \in L$ for some $j \in \{1, \ldots, t(n)\}$.

Essentially one can make mistakes with probability smaller than ξ but otherwise the algorithm should produce a conjunctive OR-distillation. And analogously we can define probabilistic disjunctive AND-distillations.

▶ **Definition 23** (Probabilistic disjunctive AND-distillation)**.** Let $L, R \subseteq \{0,1\}^*$ be a pair of languages and let $t\colon \mathbb{N} \to \mathbb{N} \setminus \{0\}$ be a function. Then a probabilistic t-bounded disjunctive AND-distillation algorithm from L into R with bounded error $0 \leq \xi < 0.5$ is an algorithm A that for every n, given $t(n)$ input strings $x_1, \ldots, x_{t(n)}$, with $|x_i| = n$ for all i, A runs in polynomial time and outputs $t(n)$ strings $y_1, \ldots, y_{t(n)}$, where each y_i has length at most $t(n) \log n$, such that with probability greater than $1 - \xi$ $y_i \in R$ for some $i \in \{1, \ldots, t(n)\}$ if and only if $x_j \in L$ for every $j \in \{1, \ldots, t(n)\}$.

We can now prove that if such distillations exist then the language is in coNP/poly by adapting the proof by Drucker [10] to work as well for the probabilistic version of conjunctive and disjunctive distillations.

▶ **Theorem 24.** *Let L be any language. Suppose that $t(n)\colon \mathbb{N}^+ \to \mathbb{N}^+$ is a function. Suppose that there exists a probabilistic conjunctive OR-distillation or a probabilistic disjunctive AND-distillation defined by $p(n)$ functions $f_i\colon \{0,1\}^{t(n)\times n} \to \{0,1\}^{t(n)\log n}$ for L with parameter $t(n)$, error bound $\xi(n) < 0.5$, and some target language R. Let*

$$\hat{\delta} = \min\left\{\sqrt{(\ln 2)(t'+1)/2t}, 1 - 2^{-3-t'/t}\right\}.$$

If, for some constant $c > 0$, we have $1 - 2\xi(n) - \hat{\delta} \geq 1/n^c$, then $L \in$ coNP/poly.

The proof of this theorem is completely analogous to the proof of Theorem 16, where the treatment of the probabilistic aspect is analogous to the one by Drucker [10]. One needs to be careful to state an appropriate version of Claim 17 for both the conjunctive and disjunctive cases, which should take into account the probabilistic aspect of the distillation, the OR or AND properties of the reduction itself, as well as the conjunctive or disjunctive aspect. These claims read as follows.

▷ Claim 25. If L is a language with a probabilistic conjunctive OR-distillation as determined in Theorem 24, the following holds:

1. If $y \notin L$, then there exists an i such that, $\|\mathcal{D}_{C_i} - \mathcal{D}_{C'_i}\|_{\text{stat}} \geq D(n) = 1 - 2\xi(n)$;
2. If $y \in L$ then for every i, we have $\|\mathcal{D}_{C_i} - \mathcal{D}_{C'_i}\|_{\text{stat}} \leq d(n) = \hat{\delta} + 1/2n^c$

▷ Claim 26. If L is a language with a probabilistic disjunctive AND-distillation as determined in Theorem 24, the following holds:

1. If $y \notin L$, then for every i, we have $\|\mathcal{D}_{C_i} - \mathcal{D}_{C'_i}\|_{\text{stat}} \geq D(n) = 1 - 2\xi(n)$;
2. If $y \in L$ then there exists an i such that, $\|\mathcal{D}_{C_i} - \mathcal{D}_{C'_i}\|_{\text{stat}} \leq d(n) = \hat{\delta} + 1/2n^c$

These claims can be proven using Lemma 14 and then Theorem 24 follows also from applying the same circuit reduction, now with the probabilistic aspect, and using Theorem 15.

4.2 XOR-compositional Problems

In the same way that we defined for a problem L, the problem OR(L) and AND(L) one can easily imagine defining problems for other boolean operations, for example XOR(L) would count the parity of the number of yes instances. Formally,

▶ **Definition 27.** Given a problem L, we define the boolean variable $b_L(x)$ such that $b_L(x) = 1$ if and only if $x \in L$. We say that a set of instances $(x_1, \ldots, x_t)$ each having a length of at most n is in $\text{XOR}(L)$ if and only if $\text{XOR}(b_L(x_1), \ldots, b_L(x_t)) = 1$.

Analogously we define XOR-cross-composition.

▶ **Definition 28.** Let $L \subseteq \Sigma^*$ be a language and $Q \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized language. We say that L XOR-cross-composes into Q if there exists a polynomial equivalence relation R and an algoritm A called a cross-composition such that A takes as input $x_1, \ldots, x_t \in \Sigma^*$ equivalent with respect to R, and outputs one instance $(y, k) \in \Sigma^* \times N$ such that k is polynomial in the size of x_i and $\log t$, and $(y, k) \in Q$ if and only if there is an odd number of instances $x_i \in L$.

The goal is to prove that for some problems L, if one has an XOR-cross-composition from L into a parameterized problem Q, and a conjunctive or disjunctive kernelization for Q, then one would be able to build a conjunctive or disjunctive distillation for L, which by Theorem 24 means that $L \subseteq \text{NP/poly}$. Thus, if L is for instance NP-hard, Q is unlikely to have conjunctive or disjunctive kernels.

In general, when we think of problems that XOR-cross-compose to their own parameterized versions, or problems where L is equivalent to $\text{XOR}(L)$, an instance will be in L then, if we have an odd number of solutions, which in general means, that they belong to $\oplus$P, or are $\oplus$P-hard instead of NP-hard, making it difficult to find problems that are unlikely to have conjunctive and disjunctive, or even polynomial kernels through the usual complexity-theoretic consequence coNP $\subseteq$ NP/poly. To avoid this problem, we establish a collapse of the polynomial hierarchy in the case of $\oplus$P problems. First, we formally define some complexity classes and state a few complexity theoretic results.

We first define the general complexity class $\text{BP} \cdot \mathcal{C}$.

▶ **Definition 29** (Schöning [21])**.** Given a complexity class $\mathcal{C}$, the bounded error probabilistic-$\mathcal{C}$ class ($\text{BP} \cdot \mathcal{C}$), is defined as the class of all languages L such that for some language $L' \in \mathcal{C}$, for some polynomial p and for all inputs x of length n, we have

$$\text{Prob}[(x, y) \in L' \text{ if } x \in L] \geq 2/3$$
$$\text{Prob}[(x, y) \in L' \text{ if } x \notin L] \leq 1/3$$

where y is chosen uniformly at random from all strings of length $p(n)$.

With this definition we prove the following theorem.

▶ **Theorem 30.** $\text{BP} \cdot (\text{NP/poly}) \subseteq \text{NP/poly}$.

Proof. Given the defintion of $\text{BP} \cdot (\mathcal{C})$, and using amplification, we know that a language L is in $\text{BP} \cdot (\text{NP/poly})$ if there exists a language $L' \in \text{NP/poly}$ and a polynomial p such that for all inputs of length n, we have

$$\text{Prob}[(x, y) \in L' \text{ if } x \in L] \geq 1 - 2^{-n^2}$$
$$\text{Prob}[(x, y) \in L' \text{ if } x \notin L] \leq 2^{-n^2}$$

where y is chosen uniformly at random from all strings of length $p(n)$.

Let us consider such a language $L \in \text{BP} \cdot (\text{NP/poly})$, with its corresponding language $L' \in \text{NP/poly}$ and polynomial p. There are only 2^n many inputs of length n for any given n, thus, by the union bound, the probability that a string string y provides a bad outcome for

at least one input x of length n is at most $2^n \cdot 2^{-n^2} < 1$. Thus, there must be a fixed string y of length $p(n)$, such that $x \in L$ if and only if $(x, y) \in L'$ for every input x of length n.

A nondeterministic Turing machine M with polynomial advice can decide whether $x \in L$ in polynomial time as follows. Given a nondeterministic Turing machine N with polynomial advice solving L', M obtains first y from the advice tape and then tests whether $(x, y) \in L'$ by simulating N on (x, y) by using the subsequent advice that it also obtains from the same advice tape. Observe that both the advice and the running time of M are polynomial. ◀

We now formally define $\oplus$P introduced by Papadimitriou and Zachos [19].

▶ **Definition 31.** $\oplus$P is the class of decision problems solvable by a nondeterminsitic Turing machine in polynomial time, with an odd number of accepting computation paths.

An example of a $\oplus$P problem is Odd Path, where given a graph G one has to decide whether G contains an odd number of k-paths. For the purposes of this paper, we are interested on Odd Path on planar graphs, which can be formally defined as:

Input: A planar graph G and an integer k.
Task: Decide whether G contains an odd number of k-paths.

There are many other examples of problems in $\oplus$P or problems that are complete for $\oplus$P, a compillation can be found in [23].

An important property of $\oplus$P as it relates to the polynomial hierarchy (PH) is that PH $\subseteq$ BP $\cdot \oplus$P, which is an important ingredient in the proof of Toda's theorem [22]. This property together with Theorem 30 allows us to prove that $\oplus P$-hard problems are unlikely to be in coNP/poly we can do it in the following way

▶ **Theorem 32.** *If a problem L is both $\oplus$P-hard and $L \in$ coNP/poly, then* coNP $\subseteq$ NP/poly.

Proof. If a language L is both $\oplus$P-hard and $L \in$ coNP/poly, this means that $\oplus$P $\subseteq$ coNP/poly, in particular, because $\oplus$P is closed under complement it implies that $\oplus$P $\subseteq$ NP/poly.

Using the fact that PH $\subseteq$ BP $\cdot \oplus$P [22, Theorem 3.1] and Theorem 30, we can see that $\oplus$P $\subseteq$ NP/poly implies that

$$\text{PH} \subseteq \text{BP} \cdot \oplus\text{P} \subseteq \text{BP} \cdot (\text{NP/poly}) \subseteq \text{NP/poly} ,$$

which is unlikely because it makes the polynomial hierarchy colapse. In particular, coNP $\subseteq$ PH $\subseteq$ NP/poly brings us to our usual complexity theoretic consequence for this kind of lower bounds. A result by Yap [25] shows that coNP $\subseteq$ NP/poly means that the polynomial hierarchy collapses to the third level. A more refined result by Köbler and Watanabe [16] shows a stronger collapse, that is, if coNP $\subseteq$ NP/poly then the polynomial hierarchy collapses to ZPP(Σ_2^{P}). ◀

Moreover, we are able to prove that problems which XOR-cross compose into parameterized problems that have conjunctive or disjunctive polynomial kernels are in NP/poly.

▶ **Theorem 33.** *Let $L \subseteq \Sigma^*$ be a language. If L XOR-cross-composes into a parameterized problem Q and Q has a conjunctive or disjunctive kernelization, then $L \in$* coNP/poly.

Proof. We prove for conjunctive kernelizations, as we will see, the case for disjunctive kernelizations is analogous.

Let $L \subseteq \Sigma^*$ be a language, and let A be an XOR-cross-composes into a parameterized problem Q, which has a conjunctive kernelization. Let us build a probabilistic conjunctive OR-distillation for L. Given a set of inputs $x_1, \ldots x_t$ for L, the first step in our distillation is

to delete every input x_i at random with probability 1/2. This way, if there are any $x_i \in L$, after this procedure there will be an odd number of them with probability 1/2 and if there were none to begin with, there are still none after this procedure. Then, to the remaining set of instances $x'_1, \ldots, x'_{t'}$ we apply the XOR-cross-composition and from the result we build a conjunctive kernel. The probability of success is 1 if the initial input contained no $x_i \in L$, and otherwise the probability of sucess is 1/2. This is not good enough, as we need a probabilistic conjunctive OR-distillation with an error bound smaller than 1/2 to be able to apply Theorem 24. In order to do this we sacrifice the onesided error in a clever way. With a probability of 1/4 we substitute the remaining set of instances $x'_1, \ldots, x'_{t'}$ for a single instance $x \in L$ before applying the XOR-cross-composition. The probability of success is now $3/8 + 1/4 = 5/8$ if $x_i \in L$ for some i, and 3/4 otherwise, making the total error bound $\xi \leq 3/8$. We have thus, a probabilistic conjunctive OR-distillation satisfying all of the conditions to apply Theorem 24, which means that $L \in \text{coNP/poly}$. The reduction for disjunctive kernelizations is analogous. ◀

Thus, by Theorems 32 and 33 if a problem is hard for NP or $\oplus$P and reduces to a parameterized problem, then the parameterized problem is unlikely to have polynomial conjunctive or disjunctive kernels.

▶ **Corollary 34.** *The* Odd Path *problem on planar graphs is FPT and does not have conjunctive or disjunctive kernels, unless* coNP $\subseteq$ NP/poly.

Proof. The Odd Path problem is FPT on planar graphs because it can be solved by dynamic programming on graphs of bounded tree-depth using standard techniques. Reducing the problem from planar graphs to graphs of bounded tree-depth can be done with low tree-depth colorings [18].

The problem trivially XOR-cross-composes to itself and is in $\oplus$P. Moreover, Odd Path on planar graphs is $\oplus$P-hard because it can be reduced to the problem of finding the parity of the number of Hamiltionian paths ($k = n$) on planar graphs with maximum degree 3, which is $\oplus$P-complete [23]. ◀

The question whether Odd Path on general graphs is FPT remained open until very recently, when Curticapean, Dell and Husfeldt proved that Odd Path is $\oplus$-W[1]-complete, thus is unlikely to be fixed parameter tractable [6].

4.3 The Weighted Odd Path Problem

In order to find an FPT problem without conjunctive or disjunctive kernels, with a reduction from an NP-hard problem instead of a reduction from a $\oplus$P-hard problem, we consider the Weighted Odd Path problem with parameter k on planar graphs, i.e., the weighted version of the Odd Path problem:

Input: A planar graph G with edge weights and two integers k and m.
Task: Decide whether G contains an odd number of k-paths with weight m.

This problem is in FPT on planar graphs because it can be solved by dynamic programming on graphs of bounded tree-depth using standard techniques. Reducing the problem from planar graphs to graphs of bounded tree-depth can be done with low tree-depth colorings [18].

We now prove, through a reduction from Longest Path, that Weighted Odd Path is unlikely to have conjunctive or disjunctive kernels. The reduction uses a probabilistic XOR-cross-composition and then proves that conjunctive and disjunctive kernels are still unlikely following the same techniques as used in Theorem 33.

▶ **Theorem 35.** *Unless* coNP $\subseteq$ NP/poly*,* Weighted Odd Path *on planar graphs has no disjunctive or conjunctive kernelizations.*

Proof. The XOR-cross-composition from Longest Path into Weighted Odd Path relies on the following observation. Let G be a graph of order n. If we turn G into an edge-weighted graph by equipping each edge with a random weight from $\{1, \ldots, n^3\}$ then the Isolation Lemma [17] shows that a k-path with minimum weight is unique with a failure probability of at most $1/n$. If, on the other hand, G does not contain a k-path, then its weighted version will not contain a k-path of any weight.

Thus, the reduction runs as follows. First we turn G into a weighted graph G' as explained above. Then create $k(n^3 - 1) + 1$ different instances $(G', k, k), (G', k + 1, k), \ldots, (G', kn^3, k)$ of the Weighted Odd Path problem. If G does contain a k-path, then with probability of at least $1 - 1/n$ one of the instances is a yes-instance. Second, we delete each instance from this collection independently with a probability of $1/2$. Let us call the remaining set of instances I. Now I contains an odd number of yes-instances with probabilty of at least $1/2 - 1/2n$ if G contains at least one k-path. Now, to offset the one-sided error, just as we did in the previous subsection, with a probability of $1/4$ we redefine I to be a single (trivial) yes-instance. If G contained no k-path, the probability that I does not contain a k-path is now $3/4$, whereas if G contains at least one k-path, now I contains an odd number of yes-instances with probabilty of at least $1/4 + 3/8(1 - 1/n) = 5/8 - 3/8n > 1/2$.

Third, we turn I into a single graph as follows: If I contains (G', i, k) we double each edge weight in G' and call the resulting weighted graph G''_i. Then we attach to each vertex in G''_i a new pendant vertex via a new edge with weight $n^7/2 - i$. Note that G''_i contains a $k + 2$-path of weight n^7 iff G' contains a k-path of weight i. Finally we create a new graph G''' that consists of all G''_i for $i \in \{k, \ldots, kn^3\}$.

We arrived at the following situation: If G does not contain a k-path, then with a probability of $3/4$ the graph G''' does not contain a $k + 2$-path of weight n^7. Otherwise, if G contains a k-path, G''' contains a $k + 2$-path of weight n^7 with a probability of at least $5/8 - 3/8n$.

Let us now assume that Weighted Odd Path has a polynomial conjunctive kernel. Then we can use this kernel on G''' and get a randomized conjunctive OR-distillation for Longest Path, which implies, by Theorem 24 that NP $\in$ coNP/poly. Hence, the Weighted Odd Path problem should not have a conjunctive polynomial kernel. Because there is an almost trivial self reduction to its complement it should also have no polynomial disjunctive kernel. ◀

5 Conclusion

Lower bounds were part of parameterized complexity from the very begining in terms of W-completeness. However, it took more than 20 years for the first strong bounds on non-existence of many-one polynomial kernels, which was finally achieved by the concerted efforts of research in classical and parameterized complexity theory. Today, the next big open question are similar lower bounds for polynomial Turing kernels, a task which has so far withstood every solution attempt. In this paper, we went a little beyond many-one kernels by considering disjunctive and conjunctive kernels, which are arguably the most common types of Turing kernels in parameterized algorithmics. Besides the big question about general Turing kernels we pose the following open questions: Can we prove the same lower bounds for Longest Path as we did for Odd Path and Weighted Odd Path? Can we prove the nonexistence of disjunctive kernels for any of the WK[1]-hard problems?

References

1 Daniel Binkele-Raible, Henning Fernau, Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Yngve Villanger. Kernel(s) for problems with no kernel: On out-trees with many leaves. *ACM Trans. Algorithms*, 8(4):38:1–38:19, 2012. doi:10.1145/2344422.2344428.

2 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009. doi:10.1016/j.jcss.2009.04.001.

3 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. doi:10.1137/120880240.

4 Hans L. Bodlaender, Stéphan Thomassé, and Anders Yeo. Kernel bounds for disjoint cycles and disjoint paths. *Theor. Comput. Sci.*, 412(35):4570–4578, 2011. doi:10.1016/j.tcs.2011.04.039.

5 L. Bodlaender, Hans, Erik D. Demaine, Michael R. Fellows, Jiong Guo, Danny Hermelin, Daniel Lokshtanov, Moritz Müller, Venkatesh Raman, Johan van Rooij, and Frances A. Rosamond. Open problems in parameterized and exact computation – IWPEC 2008. Technical Report UU-CS-2008-017, Dept. of Informatics and Computing Sciences, Utrecht University, 2008.

6 Radu Curticapean, Holger Dell, and Thore Husfeldt. Modular Counting of Subgraphs: Matchings, Matching-Splittable Graphs, and Paths. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms (ESA 2021)*, volume 204 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 34:1–34:17, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2021.34.

7 Marek Cygan, Marcin Pilipczuk, and Michal Pilipczuk. Known algorithms for edge clique cover are probably optimal. *SIAM J. Comput.*, 45(1):67–83, 2016. doi:10.1137/130947076.

8 Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Kernelization lower bounds through colors and ids. *ACM Trans. Algorithms*, 11(2):13:1–13:20, 2014. doi:10.1145/2650261.

9 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

10 Andrew Drucker. New limits to classical and quantum instance compression. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 609–618. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.71.

11 Andrew Drucker. New limits to classical and quantum instance compression. *SIAM J. Comput.*, 44(5):1443–1479, 2015. doi:10.1137/130927115.

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 Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *J. Comput. Syst. Sci.*, 77(1):91–106, 2011. doi:10.1016/j.jcss.2010.06.007.

14 Danny Hermelin, Stefan Kratsch, Karolina Soltys, Magnus Wahlström, and Xi Wu. A completeness theory for polynomial (Turing) kernelization. *Algorithmica*, 71(3):702–730, 2015. doi:10.1007/s00453-014-9910-8.

15 Bart M. P. Jansen. Turing kernelization for finding long paths and cycles in restricted graph classes. *J. Comput. Syst. Sci.*, 85:18–37, 2017. doi:10.1016/j.jcss.2016.10.008.

16 Johannes Köbler and Osamu Watanabe. New collapse consequences of NP having small circuits. In Zoltán Fülöp and Ferenc Gécseg, editors, *Automata, Languages and Programming, 22nd International Colloquium, ICALP95, Szeged, Hungary, July 10-14, 1995, Proceedings*, volume 944 of *Lecture Notes in Computer Science*, pages 196–207. Springer, 1995. doi:10.1007/3-540-60084-1_74.

17 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Comb.*, 7(1):105–113, 1987. doi:10.1007/BF02579206.

18 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity – Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-27875-4.

19 Christos H. Papadimitriou and Stathis Zachos. Two remarks on the power of counting. In Armin B. Cremers and Hans-Peter Kriegel, editors, *Theoretical Computer Science, 6th GI-Conference, Dortmund, Germany, January 5-7, 1983, Proceedings*, volume 145 of *Lecture Notes in Computer Science*, pages 269–276. Springer, 1983. doi:10.1007/BFb0009651.

20 Amit Sahai and Salil P. Vadhan. A complete problem for statistical zero knowledge. *J. ACM*, 50(2):196–249, 2003. doi:10.1145/636865.636868.

21 Uwe Schöning. Probabilistic complexity classes and lowness. *J. Comput. Syst. Sci.*, 39(1):84–100, 1989. doi:10.1016/0022-0000(89)90020-2.

22 Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991. doi:10.1137/0220053.

23 Leslie G. Valiant. Completeness for parity problems. In Lusheng Wang, editor, *Computing and Combinatorics, 11th Annual International Conference, COCOON 2005, Kunming, China, August 16-29, 2005, Proceedings*, volume 3595 of *Lecture Notes in Computer Science*, pages 1–8. Springer, 2005. doi:10.1007/11533719_1.

24 Jouke Witteveen, Ralph Bottesch, and Leen Torenvliet. A hierarchy of polynomial kernels. In Barbara Catania, Rastislav Královic, Jerzy R. Nawrocki, and Giovanni Pighizzini, editors, *SOFSEM 2019: Theory and Practice of Computer Science - 45th International Conference on Current Trends in Theory and Practice of Computer Science, Nový Smokovec, Slovakia, January 27-30, 2019, Proceedings*, volume 11376 of *Lecture Notes in Computer Science*, pages 504–518. Springer, 2019. doi:10.1007/978-3-030-10801-4_39.

25 Chee-Keng Yap. Some consequences of non-uniform conditions on uniform classes. *Theor. Comput. Sci.*, 26:287–300, 1983. doi:10.1016/0304-3975(83)90020-8.

Improved Kernels for Edge Modification Problems

Yixin Cao
Department of Computing, Hong Kong Polytechnic University, Hong Kong, China

Yuping Ke
Department of Computing, Hong Kong Polytechnic University, Hong Kong, China

Abstract

In an edge modification problem, we are asked to modify at most k edges of a given graph to make the graph satisfy a certain property. Depending on the operations allowed, we have the completion problems and the edge deletion problems. A great amount of efforts have been devoted to understanding the kernelization complexity of these problems. We revisit several well-studied edge modification problems, and develop improved kernels for them:

- a 2k-vertex kernel for the cluster edge deletion problem,
- a $3k^2$-vertex kernel for the trivially perfect completion problem,
- a $5k^{1.5}$-vertex kernel for the split completion problem and the split edge deletion problem, and
- a $5k^{1.5}$-vertex kernel for the pseudo-split completion problem and the pseudo-split edge deletion problem.

Moreover, our kernels for split completion and pseudo-split completion have only $O(k^{2.5})$ edges. Our results also include a 2k-vertex kernel for the strong triadic closure problem, which is related to cluster edge deletion.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Kernelization, edge modification, cluster, trivially perfect graphs, split graphs

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.13

Related Version *Full Version*: `https://arxiv.org/abs/2104.14510`

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

1 Introduction

In an edge modification problem, we are asked to modify at most k edges of a given graph G to make the graph satisfy a certain property. In particular, we have edge deletion problems and completion problems when the allowed operations are edge deletions and, respectively, edge additions. There is also a more general version that allows both operations. The present paper will be focused on a single type of modifications. For most graph properties, these edge modification problems are known to be NP-complete [21, 18, 15]. A graph G having a certain property is equivalent to that G belongs to some specific graph class. Cai [2] observed that if the desired graph class can be characterized by a finite number of forbidden induced subgraphs, then these problems are fixed-parameter tractable.

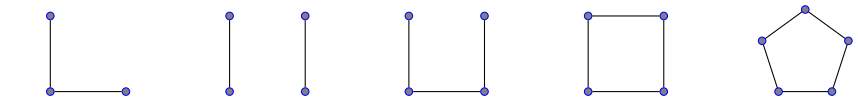
One is then naturally interested in the kernelization complexity of edge modification problems toward these *easy* graph classes. Given an instance (G, k), a *k*ernelization algorithm produces in polynomial time an equivalent instance (G', k') – (G, k) is a yes-instance if and only if (G', k') is a yes-instance – such that $k' \leqslant k$ and the size of G' is bounded by a computable function of k. The output instance (G', k') is a *polynomial kernel* if the size of G' is bounded from above by a polynomial function of k'. Although progress has been made in this regard, we get stuck for several important graph classes. We have evidence that some of them do not have polynomial kernels, under certain complexity assumptions, and it is believed that those that do have are exceptions [16]. This makes a sharp contrast with the vertex deletion problems (deleting vertices instead of edges), for which a polynomial kernel

is guaranteed when the number of forbidden induced subgraphs is finite [9]. We refer the reader to the recent survey of Crespelle et al. [4], particularly its Section 2.1 and Table 1, for the most relevant results.

We revisit several well-studied edge modification problems, and develop improved kernels for them. Our results are summarized in Table 1. All the target graph classes can be defined by a small number of forbidden induced subgraphs (listed in Figure 1). It is worth mentioning that the edge deletion problem to a graph class is polynomially equivalent to the completion problem to its complement graph class (consisting of the complements of all graphs in the original graph class). Moreover, some graph classes, e.g., split graphs ($\{2K_2, C_4, C_5\}$-free), are self-complementary, and thus the edge deletion problem and the completion problem toward such a class are equivalent.

Table 1 Main results of this paper, shown as the number of vertices in the kernels.

problem	previous result	our result
cluster edge deletion	$4k$ [11]	$2k$
trivially perfect completion	$O(k^7)$ [7]	$3k^2$
split completion (edge deletion)	$O(k^2)$ [10]	$5k^{1.5}$
pseudo-split completion (edge deletion)	-	$5k^{1.5}$
strong triadic closure	$4k$ [11]	$2k$



(a) P_3. **(b)** $2K_2$. **(c)** P_4. **(d)** C_4. **(e)** C_5.

Figure 1 Forbidden induced graphs. Note that $2K_2$ and C_4 are complements of each other, while the complements of P_4 and C_5 are themselves.

A cluster graph is a disjoint union of cliques. Since cluster graphs are precisely P_3-free graphs, edge modification problems to cluster graphs are the simplest of all nontrivial edge modification problems. Note that edge modification problems toward P_2-free graphs, i.e., edgeless graphs, are trivial. Also trivial is the cluster completion problem: the minimum solution is to add edges to make every component of the input graph complete. Both cluster edge editing and cluster edge deletion are NP-complete and have received wide attentions. After a sequence of results, Cao and Chen [3] devised a $2k$-vertex kernel for the cluster edge editing problem. Their algorithm actually implies a $2k$-vertex kernel for the cluster edge deletion problem. We record this simple result here for future reference. Less trivially, we show that the same algorithm produces a kernel of the same size for the strong triadic closure problem, which, though originally not posed as an edge modification problem, is closely related to cluster edge deletion [14]. As the original results [3], both algorithms work for the weighted versions of the problems as well.

The second problem is the trivially perfect completion problem. Drange and Pilipczuk [7] presented an $O(k^7)$-vertex kernel for this problem.[1] We propose a very simple kernelization algorithm, which has only two simple reduction rules, and the resulting kernel contains at

[1] Guo [12] has claimed an $O(k^3)$-vertex kernel for this problem, with details deferred to a full version that has never appeared. The algorithm of Drange and Pilipczuk [7] works for the more general trivially perfect editing problem. Independent to our work, Dumas et al. [8] improved it to $O(k^3)$, which also implies an $O(k^3)$-vertex kernel for the trivially perfect completion problem.

most $2k^2 + 2k$ vertices. The forbidden induced subgraphs of trivially perfect graphs are P_4 and C_4. Note that adding the edge to connect the two ends of a P_4 merely turns it into a C_4. Thus, in each P_4 or C_4, there are two missing edges such that every solution needs to contain at least one of them. Note that each vertex of the P_4 or C_4 is an end of one of the two missing edges. Our first rule is the most routine for this kind of problems, namely, adding a missing edge if it is one of the two possible missing edges in $k + 1$ or more P_4's and C_4's. Our second rule removes all vertices that are not contained in any P_4 or C_4 of G. Now the analysis is similar as Buss and Goldsmith's kernelization algorithm for the vertex cover problem [1]. Since every solution contains at least one of the pair of potential missing edges (for some P_4 or C_4), and since each potential edge is in at most k pairs, there cannot be more than $k^2 + k$ potential edges in a yes-instance. On the other hand, every vertex is in a P_4 or C_4, hence an end of some potential edge. We are thus safe to return a trivial no-instance when $|V(G)| > 2k^2 + 2k$. Toward this result we also obtain some nontrivial observations on minimal solutions of the problem with respect to modules of the input graph.

A graph is a split graph if its vertex set can be partitioned into a clique and an independent set. Split graphs are $\{2K_2, C_4, C_5\}$-free graphs. The split completion problem, which is equivalent to split edge deletion, is NP-complete [17], while somewhat surprisingly, the split edge editing problem can be solved in polynomial time [13]. Guo [12] presented an $O(k^4)$-vertex kernel for the split completion problem, which was improved to $O(k^2)$ by Ghosh et al. [10]. For the convenience of presentation, we work on the edge deletion problem. We consider the partition of the vertex set after applying an optimal solution. We observe that for most of the vertices we know to which side they have to belong. It is nevertheless not safe to directly delete these "decided" vertices. We thus work on the annotated version, where we mark certain vertices that have to be in the independent set. Guo [12] has proved that it is safe to remove a vertex that is not contained in any $2K_2$, C_4, or C_5. We show that a similar rule can be applied to annotated instances, and after its application, there can be at most $O(k^{1.5})$ vertices in a yes-instance. Finally, a simple step that removes the marks concludes the algorithm. Our kernel for split completion has only $O(k^{2.5})$ edges.[2] With minor tweaks, our algorithm produces a kernel of the same size for the pseudo-split ($\{2K_2, C_4\}$-free graphs) edge deletion problem. A pseudo-split graph is either a split graph or a split graph plus a C_5 such that every vertex on the C_5 is adjacent to every vertex in the clique part of the split graph and is nonadjacent to any vertex in the independent part of the split graph. The first difficulty toward this adaptation is that it is not always safe to remove vertices not contained in any $2K_2$ or C_4. We get over this obstacle by observing that we can remove vertices not contained in any $2K_2$, C_4, or C_5. As we recycle the reduction rules for split edge deletion, only the arguments for their safeness need to be slightly revised. Drange et al. [6] have used a similar approach to develop a subexponential-time algorithm for the pseudo-split completion problem.

2 Preliminaries

All graphs discussed in this paper are undirected and simple. The vertex set and edge set of a graph G are denoted by, respectively, $V(G)$ and $E(G)$. For a subset $U \subseteq V(G)$, denote by $G[U]$ the subgraph of G induced by U, and by $G - U$ the subgraph $G[V(G) \setminus U]$, which is further shortened to $G - v$ when $U = \{v\}$. The *neighborhood* of a vertex v in G, denoted

[2] Independent to our work, Bathie, Bousquet, and Pierron (arXiv:2105.09566) use a similar technique to obtain a linear-vertex kernel for these problems.

by $N_G(v)$, comprises vertices adjacent to v, i.e., $N_G(v) = \{u \mid uv \in E(G)\}$, and the *closed neighborhood* of v is $N_G[v] = N_G(v) \cup \{v\}$. The *closed neighborhood* and the *neighborhood* of a set $U \subseteq V(G)$ of vertices are defined as $N_G[U] = \bigcup_{v \in U} N_G[v]$ and $N_G(U) = N_G[U] \setminus U$, respectively. We may omit the subscript when there is no ambiguity on the graph under discussion. Two vertices u and v are true twins in G if $N[u] = N[v]$; note that true twins are necessarily adjacent. A *clique* is a set of pairwise adjacent vertices, and an *independent set* is a set of pairwise nonadjacent vertices. A graph G is *complete* if $V(G)$ is a clique. A vertex v is *simplicial* if $N[v]$ is a clique, and a vertex v is *universal* if $N[v] = V(G)$. An induced path and an induced cycle on ℓ vertices are denoted by P_ℓ and C_ℓ respectively.

For any two subsets $X, Y \subseteq V(G)$, we use $E(X, Y)$ to denote the set of edges of which one endpoint is in X and the other in Y. Note that we do not require X and Y to be disjoint. Thus, $E(X, X) = E(G[X])$, i.e., all the edges with both endpoints in X, and $E(X, V(G))$ consists of all the edges with at least one endpoint in X.

Let F be a fixed graph. We say that a graph G is F-*free* if G does not contain F as an induced subgraph. For a set $\mathcal{F}$ of graphs, a graph G is $\mathcal{F}$-*free* if G is F-free for every $F \in \mathcal{F}$. If every $F \in \mathcal{F}$ is minimal, i.e., not containing any $F' \in \mathcal{F}$ as a proper induced subgraph, then the set $\mathcal{F}$ of graphs are the (minimal) *forbidden induced subgraphs* of this class. See Figure 1 for the forbidden induced subgraphs considered in the present paper. For a set E' of non-edges, we denote by $G + E'$ the graph with vertex set $V(G)$ and edge set $E(G) \cup E'$; for a set $E' \subseteq E(G)$, we denote by $G - E'$ the graph with vertex set $V(G)$ and edge set $E(G) \setminus E'$. The problems to be studied are formally defined as follows, where $\mathcal{G}$ is a graph class.

$\mathcal{G}$ completion

Input: A graph G and a nonnegative integer k.
Output: Is there a set E_+ of at most k edges such that $G + E_+$ is in $\mathcal{G}$?

$\mathcal{G}$ edge deletion

Input: A graph G and a nonnegative integer k.
Output: Is there a set E_- of at most k edges such that $G - E_-$ is in $\mathcal{G}$?

Since it is always clear from the context what problem we are talking about, when we mention an instance (G, k), we do not always explicitly specify the problem. We use $\mathrm{opt}(G)$ to denote the size of optimal solutions of G for the optimization version of a certain problem. Thus, (G, k) is a yes-instance if and only if $\mathrm{opt}(G) \leqslant k$.

For each problem, we apply a sequence of reduction rules. Each rule transforms an instance (G, k) to a new instance (G', k'). We say that a rule is *safe* if (G, k) is a yes-instance if and only if (G', k') is a yes-instance. Since all of our reduction rules are very simple and most of them are obviously doable in polynomial time, we omit the details of their implementation and analyze their running time only when it is nontrivial.

3 Cluster edge deletion and strong triadic closure

A graph is a cluster graph if every component of this graph is a complete subgraph. It is well known that a graph is a cluster graph if and only if it is P_3-free. Our first problem is the cluster edge deletion problem. For a vertex set $U \subseteq V(G)$, we write $d(U) = |E(U, V(G) \setminus U)|$, i.e., the number of edges between U and $V(G) \setminus U$; we write $d(v)$ instead of $d(\{v\})$ for a singleton set.

▶ **Rule 3.1.** *If there is a simplicial vertex* v *such that* $d(N[v]) \leqslant d(v)$*, then remove* $N[v]$ *and decrease* k *by* $d(N[v])$*.*

Safeness of Rule 3.1. We show that $\mathrm{opt}(G) = \mathrm{opt}(G - N[v]) + d(N[v])$. Let E_- be an optimal solution to the graph G. We have nothing to show if $N[v]$ is a separate component of $G - E_-$. In the rest of the proof, $N[v]$ is not a component of $G - E_-$. Let $G[X]$ denote the component of $G - E_-$ that contains v. Since X is a clique and $N[v] \neq X$, we have $X \subset N[v]$. In other words, neither X nor $N[v] \setminus X$ is empty. Since any induced subgraph of $G - E_-$ is a cluster graph, the subset of edges in E_- with both endpoints in $V(G) \setminus N[v]$ is a solution to $G - N[v]$. Noting that this solution is disjoint from $E(X, V(G) \setminus X)$, we have

$$\begin{aligned}\mathrm{opt}(G) \geqslant & |E_- \cap E(G - N[v])| + d(X)\\ \geqslant & \mathrm{opt}(G - N[v]) + |X| \cdot |N[v] \setminus X|\\ \geqslant & \mathrm{opt}(G - N[v]) + |X| + |N[v] \setminus X| - 1\\ = & \mathrm{opt}(G - N[v]) + d(v),\end{aligned} \tag{1}$$

where the third inequality holds because both $|X|$ and $|N[v] \setminus X|$ are positive integers. For any solution E'_- of $G - N[v]$, the set $E'_- \cup E(N[v], V(G) \setminus N[v])$ is a solution of G. Thus,

$$\mathrm{opt}(G) \leqslant \mathrm{opt}(G - N[v]) + d(N[v]) \leqslant \mathrm{opt}(G - N[v]) + d(v). \tag{2}$$

Therefore, all the inequalities in (1) and (2) are tight. In other words, if we remove all the edges between $N[v]$ and $V(G) \setminus N[v]$, and then delete an optimal solution to $G - N[v]$, then we have an optimal solution to the graph G. ◀

A trivial but crucial fact is that a solution E_- has at most $2|E_-|$ endpoints. If a vertex v is not an end of any edge in E_-, then v has to be simplicial.

▶ **Theorem 1.** *There is a* $2k$*-vertex kernel for the cluster edge deletion problem.*

Proof. Let G be a graph to which Rule 3.1 is not applicable. We show that if (G, k) is a yes-instance, then $|V(G)| \leqslant 2k$. Let E_- be an optimal solution to G, and let $\{v_1, v_2, \ldots, v_r\}$ be the vertices that are not incident to any edge in E_-; they have to be simplicial. For $i = 1, \ldots, r$, the set $N[v_i]$ forms a component of $G - E_-$. Note that for distinct $i, j \in \{1, \ldots, r\}$, the sets $N[v_i]$ and $N[v_j]$ are either the same (when v_i and v_j are true twins) or mutually disjoint: if $N[v_i] \neq N[v_j]$ and there exists $x \in N[v_i] \cap N[v_j]$, then one of xv_i and xv_j needs to be in E_-. We divide the cost of each edge $uv \in E_-$ and assign them to u and v equally. For $i = 1, \ldots, r$, the total cost attributed to all the vertices in $N[v_i]$ is $d(N[v_i])/2$. Each of the vertices not in $\bigcup_{i=1}^{r} N[v_i]$ is an end of at least one edge in E_- and therefore bears cost at least $1/2$. Summing them up, we get a lower bound for the total cost:

$$\begin{aligned}|E_-| \geqslant & \frac{1}{2}\sum_{i=1}^{r} d(N[v_i]) + \frac{1}{2}|V(G) \setminus \bigcup_{i=1}^{r} N[v_i]|\\ \geqslant & \frac{1}{2}\sum_{i=1}^{r} |N[v_i]| + \frac{1}{2}|V(G) \setminus \bigcup_{i=1}^{r} N[v_i]|\\ \geqslant & \frac{1}{2}|V(G)|.\end{aligned}$$

The second inequality holds because Rule 3.1 does not apply to v_i for $i = 1, \ldots, r$. Thus, $|V(G)|/2 \leqslant |E_-| \leqslant k$ for a yes-instance, and we can return a trivial no-instance if $|V(G)| > 2k$. This concludes the proof. ◀

Let us mention that the condition of Rule 3.1 can be weakened to $d(N[v]) < 2d(v) - 1$. We do not prove the stronger statement because it does not improve the analysis of the kernel size, but let us briefly explain why it is true. The bound $\mathrm{opt}(G) \geqslant \mathrm{opt}(G - N[v]) + 2d(v) - 1$ holds unless $|X| = 1$ or $|N[v] \setminus X| = 1$; see the third inequality of (1). In the first case, v itself makes a trivial component, and all the vertices in $N(v)$ are in the same component; this can only happen when there exists another vertex u with $N(v) \subseteq N(u)$. In the second case, a vertex $u \in N(v)$ is incident to all the edges between $N(v)$ and $V(G) \setminus N[v]$. If $d(N[v]) < 2d(v) - 1$, then $\mathrm{opt}(G) \geqslant \mathrm{opt}(G - N[v]) + 2d(v) - 1$ holds in both cases.

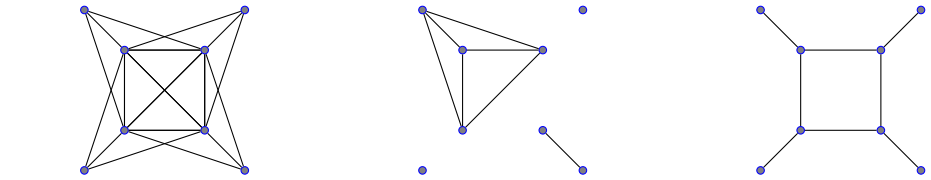


Figure 2 The example given by Konstantinidis et al. [14]: (a) the input graph; (b) a maximum cluster subgraph with seven edges; and (c) a maximum strong triadic closure with eight edges.

In the original definition, which was motivated by applications in social networks, the *strong triadic closure* problem asks for a partition of the edge set of the input graph into strong edges and weak ones, such that for every two vertices that are linked to a common neighbor with strong edges are adjacent. The objective is to maximize the number of strong edges. For our purpose, it is more convenient to define the problem as follows.

Strong triadic closure

Input:	A graph G and a nonnegative integer k.
Output:	Is there a set E_- of at most k edges such that the missing edge of every P_3 of $G - E_-$ is in $E(G)$?

Thus, we call the set of weak edges as the solution to the strong triadic closure problem. For any set $E_- \subseteq E(G)$, if $G - E_-$ is a cluster graph, then E_- is also a solution to the strong triadic closure problem: setting all edges in E_- weak, and all other edges strong is a feasible partition of $E(G)$. As illustration in Figure 2, however, a strong triadic closure of a graph can have fewer weak edges than an optimal solution to the cluster edge deletion problem on the same graph. Surprisingly, Rule 3.1 works for the strong triadic closure problem without change.

▶ **Lemma 2.** *Rule 3.1 is safe for the strong triadic closure problem.*

Proof. We show that $\mathrm{opt}(G) = \mathrm{opt}(G - N[v]) + d(N[v])$. Let E_- be an optimal solution to the graph G. We have nothing to show if $N[v]$ is a separate component of $G - E_-$. In the rest of the proof, $N[v]$ is not a component of $G - E_-$. Let X denote the set of vertices with $N[X] = N[v]$, and $Y \subseteq N[v]$ the endpoints of these edges in $E(N[v], V(G) \setminus N[v]) \setminus E_-$ (i.e., edges between $N[v]$ and $V(G) \setminus N[v]$ that are not in E_-). Note that $X \neq \emptyset$ because $v \in X$, and $Y \neq \emptyset$ because $E(N[v], V(G) \setminus N[v]) \not\subseteq E_-$ (otherwise $N[v]$ is a component of $G - E_-$ by the minimality of E_-).

By definition, the subset of edges in E_- with both endpoints in $G - N[v]$ is a solution to $G - N[v]$. By the selection of X and Y, every vertex in $N[v] \setminus (X \cup Y)$ is incident to at least one edge in $E_- \cap E(N[v], V(G) \setminus N[v])$. For every $x \in X$ and every $y \in Y$, there exists

$z \in V(G) \setminus N[v]$ that is adjacent to y but not x; hence, xyz is a P_3. As a result, all the edges between X and Y have to be in E_-. Thus,

$$\begin{aligned}\mathrm{opt}(G) =& |E_- \cap E(G - N[v])| + |E_- \cap E(N[v], V(G) \setminus N[v])| + |E_- \cap E(N[v])| \\ \geqslant& \mathrm{opt}(G - N[v]) + |N[v] \setminus (X \cup Y)| + |X| \cdot |Y| \\ \geqslant& \mathrm{opt}(G - N[v]) + |N[v]| - |X| - |Y| + |X| + |Y| - 1 \\ \geqslant& \mathrm{opt}(G - N[v]) + |N[v]| - 1 \\ =& \mathrm{opt}(G - N[v]) + d(v),\end{aligned} \tag{3}$$

where $|X| \cdot |Y| \geqslant |X| + |Y| - 1$ because both $|X|$ and $|Y|$ are positive integers. For any solution E'_- of $G - N[v]$, the set $E'_- \cup E(N[v], V(G) \setminus N[v])$ is a solution of G. Thus,

$$\mathrm{opt}(G) \leqslant \mathrm{opt}(G - N[v]) + d(N[v]) \leqslant \mathrm{opt}(G - N[v]) + d(v). \tag{4}$$

Therefore, all the inequalities in (3) and (4) are tight. In other words, if we remove all the edges between $N[v]$ and $V(G) \setminus N[v]$, and then delete an optimal solution to $G - N[v]$, then we have an optimal solution to the graph G. ◀

The proof of the following theorem is a word-for-word copy of that for Theorem 1, hence omitted.

▶ **Theorem 3.** *There is a $2k$-vertex kernel for the strong triadic closure problem.*

We should remark that our kernelization algorithms for the cluster edge deletion problem and the strong triadic closure problem work for the weighted versions as well; see [3].

4 Trivially perfect completion

In this section we study the trivially perfect completion problem. Trivially perfect graphs are $\{P_4, C_4\}$-free graphs. If there is a pair of adjacent vertices u, v such that neither $N[u] \setminus N[v]$ nor $N[v] \setminus N[u]$ is empty, then they are contained in a P_4 or C_4. Trivially perfect graphs have many nice characterizations. Here are two of them.

▶ **Theorem 4** ([19, 20])**.** *The following are equivalent for a graph H.*

- **i)** *H is a trivially perfect graph.*
- **ii)** *Every connected induced subgraph of H contains a universal vertex.*
- **iii)** *For every pair of adjacent vertices u and v, one of $N[u]$ and $N[v]$ is a subset of the other.*

If a vertex v is not contained in any P_4 or C_4, then for every neighbor u of v, one of $N[u]$ and $N[v]$ is a subset of the other.

▶ **Lemma 5** ($\star$[3])**.** *If a vertex v is not contained in any P_4 or C_4, then $\mathrm{opt}(G - v) = \mathrm{opt}(G)$.*

As a simple result of Lemma 5, we have the following reduction rule (which was mentioned by Guo [12], without a proof). In particular, all universal vertices of every component of G can be removed.

▶ **Rule 4.1.** *If there is a vertex v that is not contained in any P_4 or C_4, then remove v.*

[3] Proofs of propositions marked with $\star$ are deferred to the full version.

For each induced 4-path or 4-cycle $v_1v_2v_3v_4$, we call the missing edges $\{v_1, v_3\}$ and $\{v_2, v_4\}$ the *candidate edges* for this path or cycle. Clearly, any solution of a graph G contains at least one candidate edge of every P_4 or C_4; note that a P_4 has another missing edge, the addition of which merely turns the P_4 into a C_4.

▶ **Rule 4.2.** *If uv is a candidate edge of $k+1$ or more P_4's and C_4's in G, then add the edge uv and decrease k by one.*

Safeness of Rule 4.2. Since each P_4 or C_4 of G has precisely two candidate edges, if a solution E_+ of G does not contain uv, then E_+ must contain the other candidate edge of each of the $k+1$ P_4's and C_4's, hence $|E_+| > k$. ◀

We are thus ready for the main result of this section.

▶ **Theorem 6.** *There is a $(2k^2+2k)$-vertex kernel for the trivially perfect completion problem.*

Proof. After applying Rule 4.2 and then Rule 4.1 exhaustively, we return (G, k) if $|V(G)| \leqslant 2k^2 + 2k$, or a trivial no-instance otherwise. We consider all the candidate edges of G. We say that two candidate edges are associated if they belong to the same P_4 or C_4; i.e., their endpoints are disjoint and together induce a P_4 or C_4. Since Rule 4.2 is not applicable, each candidate edge is associated with at most k candidate edges. On the other hand, of any two associated edges, one has to be in any solution of G. Thus, if (G, k) is a yes-instance, there can be at most $k^2 + k$ candidate edges. Since Rule 4.1 is not applicable, every vertex is in some P_4 or C_4, and hence is an end of a candidate edge. Thus, $|V(G)| \leqslant 2k^2 + 2k$ if (G, k) is a yes-instance. ◀

The analysis of the kernel in Theorem 6 is essentially the same as Buss and Goldsmith's kernelization algorithm for the vertex cover problem [1]. In a sense, we are looking for a vertex cover of an auxiliary graph in which each vertex corresponds to a candidate edge of G, and two vertices are adjacent if their corresponding edges are associated. We note that the same approach implies a simple $O(k^2)$-vertex kernel for the threshold completion problem, matching the result of Drange et al. [5]. The forbidden induced subgraphs of threshold graphs are $2K_2$, P_4, and C_4. The observation on the missing edges of a P_4 or C_4 is the same as above, while the four missing edges of a $2K_2$ can be organized as two pairs such that each solution has to contain at least one from each pair. However, we are not able to employ the $2k$-vertex kernels for vertex cover to directly derive a linear-vertex kernel for either of the two problems.

Before closing this section, let us mention some observations that might be of independent interest. A set M of vertices is a module if $N(M) = N(v) \setminus M$ for every $v \in M$.

▶ **Lemma 7** ($\star$). *A module M of a graph G remains a module in any minimal trivially perfect completion $\widehat{G}$ of G.*

5 Split edge deletion and split completion

A graph is a *split graph* if its vertex set can be partitioned into a clique and an independent set. We use $C \uplus I$, where C being a clique and I an independent set, to denote a *split partition* of a split graph. Note that a split graph may have more than one split partition; e.g., a complete graph on n vertices has $n+1$ different split partitions. The forbidden induced subgraphs of split graphs are $2K_2$, C_4, and C_5. From both the definition and the forbidden induced subgraphs we can see that the complement of a split graph is also a split graph. Thus, the split completion problem is polynomially equivalent to the split edge deletion problem. For the convenience of presentation, we work on the edge deletion problem.

Note that (G, k) is a yes-instance if and only if there exists a partition $C \uplus I$ of $V(G)$ such that C is a clique and $|E(I, I)| \leqslant k$; this is a split partition of $G - E(I, I)$. We call such a partition a *valid partition* of the instance (G, k). The problem is thus equivalent to finding a valid partition. We notice that some vertices can be easily decided to which side of a valid partition they should belong. For example, unless the instance is trivial, a simplicial vertex always belong to the independent set in any valid partition. Even after we know the destinations of these vertices, however, we cannot safely delete them. This brings us to the *annotated version* of the problem, where we mark certain vertices that can only be put into the independent set in a valid partition. We use (G, I_0, k) to denote such an annotated instance, where I_0 denotes the set of marked vertices. The original instance can be viewed as $(G, \emptyset, k)$, and a valid partition of an annotated instance (G, I_0, k) needs to satisfy the additional requirement that $I_0 \subseteq I$.

We can easily retrieve back an unannotated instance from an annotated instance. It suffices to add a small number of new vertices and make each of them adjacent to all other vertices but I_0.

▶ **Rule 5.1.** *Let (G, I_0, k) be an annotated instance. Add a clique of $\sqrt{2k} + 1$ new vertices, and make each of them adjacent to all the vertices in $V(G) \setminus I_0$. Return the result as an unannotated instance.*

Safeness of Rule 5.1. Let K denote the clique of new vertices, and let (G', k) be the resulting instance. For any valid partition $C \uplus I$ of (G, I_0, k), the partition $(C \cup K) \uplus I$ is a valid partition of (G', k) because $C \subseteq V(G) \setminus I \subseteq N(x)$ for every $x \in K$. For a valid partition $C \uplus I$ of (G', k), if any vertex in I_0 is in C, then we must have $K \subseteq I$. Since K is a clique of order $\sqrt{2k} + 1$, we have $|E(I, I)| > k$, which contradicts the validity of the partition. ◀

The aforementioned observation on simplicial vertices is formalized by the following rule.

▶ **Rule 5.2.** *Let v be a simplicial vertex in $V(G) \setminus I_0$. If $|E(G - (N[v] \setminus I_0))| \leqslant k$, then return a trivial yes-instance. Otherwise, add v to I_0.*

Safeness of Rule 5.2. In the first case, $(N[v] \setminus I_0) \uplus (V(G) \setminus N[v] \cup I_0)$ is a valid partition. Otherwise, we show by contradiction that $v \in I$ in any valid partition $C \uplus I$ of (G, I_0, k). Since C is a clique, if $v \in C$, then $C \subseteq N[v] \setminus I_0$. Thus, $E(G - (N[v] \setminus I_0)) \subseteq E(I, I)$, but then $|E(I, I)| > k$, contradicting the validity of the partition. ◀

We construct a modulator M as follows. We greedily find a maximal packing of vertex-disjoint $2K_2$'s, C_4's, and C_5's. Let M be the set of vertices in all subgraphs we found. We can terminate the algorithm by returning a trivial no-instance if we have found more than k vertex-disjoint forbidden induced subgraphs from G. Henceforth, we may assume that $|M| \leqslant 5k$, and we fix a split partition $C_M \uplus I_M$ of $G - M$. The following simple observation enables us to know the destinations of more vertices.

▶ **Lemma 8** ($\star$). *For any valid partition $C \uplus I$ of (G, k), if one exists, (i) $|I_M \cap C| \leqslant 1$; and (ii) $|C_M \cap I| \leqslant \sqrt{2k}$.*

We say that a vertex is a *c-vertex*, respectively, an *i-vertex*, if it is in C, respectively, in I, for any valid partition $C \uplus I$ of (G, I_0, k). By Lemma 8(i), C contains at most one vertex in I_M. Thus, every vertex that has more than $k + 1$ neighbors in I_M is a c-vertex, while the following are i-vertices:

- every vertex with more than $\sqrt{2k}$ non-neighbors in C_M; and
- every vertex nonadjacent to a c-vertex.

We can indeed delete all the c-vertices, as long as we keep their non-neighbors marked. Note that after obtaining the initial split partition $C_M \uplus I_M$ of $G - M$, we do not need to maintain the invariant that M is a modulator, though we do maintain that C_M is a clique and that I_M is an independent set throughout. During our algorithm, we maintain M, C_M, I_M, and I_0 as a partition of $V(G)$. Therefore, whenever we mark a vertex, we remove it from the set that originally contains it, and move it to I_0.

▶ **Rule 5.3.** *Let* (G, I_0, k) *be an annotated instance.*

i) *Mark every vertex that has more than* $\sqrt{2k}$ *non-neighbors in* C_M*.*

ii) *If a vertex* v *has more than* $k+1$ *neighbors in* $I_M \cup I_0$*, then mark every vertex in* $V(G) \setminus N[v]$ *and delete* v*.*

Safeness of Rule 5.3. Let I_0' denote the set of marked vertices after the reduction. It is trivial that if the resulting instance of i) is a yes-instance, then the original is also a yes-instance. For ii), any valid partition $C' \uplus I'$ of $(G - v, I_0', k)$ can be extended to a valid partition $(C' \cup \{v\}) \uplus I'$ of (G, I_0, k) because $C' \subseteq V(G) \setminus I_0' \subseteq N[v]$.

For the other direction, let $C \uplus I$ be any valid partition of (G, I_0, k). i) Since C is a clique, $C_M \setminus N(v) \subseteq I$ for every $v \in C$. By Lemma 8(ii), if $|C_M \setminus N(v)| > \sqrt{2k}$ for some vertex v, then v has to be in I. Thus, $C \uplus I$ is also a valid partition of the new instance (G, I_0', k). ii) By Lemma 8(i), $|I_M \setminus I| \leqslant 1$. As $I_0 \subseteq I$ and $|N(v) \cap (I_M \cup I_0)| > k+1$, there are at least $k+1$ edges between v and I. Since $|E(I, I)| \leqslant k$, we must have $v \in C$. Moreover, since C is a clique, $C \subseteq N[v]$, and every vertex nonadjacent to v has to be in I. This justifies the marking of $V(G) \setminus N[v]$. Clearly, $(C \setminus \{v\}) \uplus I$ is a valid partition of $(G - v, I_0', k)$. ◀

The next rule is straightforward: since I_0 has to be in the independent set, every solution contains all the edges in $E(I_0, I_0)$.

▶ **Rule 5.4.** *Let* (G, I_0, k) *be an annotated instance. Remove all the edges in* $E(I_0, I_0)$*, and decrease* k *accordingly.*

Safeness of Rule 5.4. By the definition of the annotated instance, any solution E_- of (G, I_0, k) contains all the edges in $E(I_0, I_0)$. Moreover, $E_- \setminus E(I_0, I_0)$ is a solution to $G - E(I_0, I_0)$, and its size is at most $k - |E(I_0, I_0)|$. On the other hand, if $(G - E(I_0, I_0), k - |E(I_0, I_0)|)$ is a yes-instance, then any solution of this instance, together with $E(I_0, I_0)$, makes a solution of (G, I_0, k) of size at most k. ◀

Once there are no edges among vertices in I_0, we can replace I_0 with another independent set as long as we keep track of the number of edges between every vertex $v \in V(G) \setminus I_0$ and I_0. The following rule reduces the cardinality of I_0. Note that if Rule 5.3 is not applicable, then $p \leqslant k$.

▶ **Rule 5.5.** *Let* (G, I_0, k) *be an annotated instance where* I_0 *is an independent set. Introduce* p *new vertices* $v_1, v_2, \ldots, v_p$*, where* $p = \max_{v \in V(G)} |N(v) \cap I_0|$*. For each vertex* $x \in N(I_0)$*, make* x *adjacent to* $v_1, \ldots, v_{|N(x) \cap I_0|}$*. Remove all vertices in* I_0*, and mark the set of new vertices.*

Instead of proving the safeness of Rule 5.5, we prove a stronger statement.

▶ **Lemma 9.** *Let* (G, I_0, k) *and* (G', I_0', k) *be two annotated instances where* $G - I_0 = G' - I_0'$ *and both* I_0 *and* I_0' *are independent sets. If* $|N_G(x) \cap I_0| = |N_{G'}(x) \cap I_0'|$ *for every* $x \in V(G) \setminus I_0$*, then* (G, I_0, k) *is a yes-instance if and only if* (G', I_0', k) *is a yes-instance.*

Proof. We show that $C \uplus I$ is a valid partition of (G, I_0, k) if and only if $C \uplus ((I \setminus I_0) \cup I_0')$ is a valid partition of (G', I_0', k). Note that

$$|E(I \setminus I_0, I_0)| = \sum_{x \in I \setminus I_0} |N_G(x) \cap I_0| = \sum_{x \in I \setminus I_0'} |N_{G'}(x) \cap I_0'| = |E(I \setminus I_0', I_0')|.$$

Since $G - I_0 = G' - I_0'$, and since there is no edge in $G[I_0]$ or $G'[I_0']$, the claim follows. ◀

Let us recall an important observation of Guo [12].

▶ **Lemma 10** ([12]). *If a vertex v is not contained in any $2K_2$, C_4, or C_5, then* $\mathrm{opt}(G - v) = \mathrm{opt}(G)$.

Both Guo [12] and Ghosh et al. [10] used a rule derived from this observation to delete vertices, and this is their only rule that removes vertices from the graph. We may show that the same rule indeed works for our annotated instances, for which however we have to go through the original argument of [12]. We note that if a vertex v in I_0 is adjacent to two vertices u and w with $uw \notin E(G)$, then any solution has to contain at least one of edges uv and vw (u and w cannot be both in the clique). We say that an induced P_3 is I_0-*centered* if the degree-two vertex of this P_3 is from I_0. In a sense, I_0-centered P_3's are "minimal forbidden structures" for our annotated instances. Accordingly, a C_4 or C_5 involving a vertex from I_0 is no longer minimal. In summary, the "minimal forbidden structures" are C_4's and C_5's in $G - I_0$, all $2K_2$'s, and I_0-centered P_3's. Note that a "minimal forbidden structure" intersecting I_0 has to be a $2K_2$ or an I_0-centered P_3, and this gives another explanation of the correctness of Lemma 9, which exchanges these two kinds of "minimal forbidden structures" with each other. The following rule can be viewed as the annotated version of the rule of Guo [12], and its safeness can be argued using Lemma 10.

▶ **Rule 5.6** ($\star$). *Let (G, I_0, k) be an annotated instance where I_0 is an independent set, and let v be a vertex in C_M. If v is not contained in any $2K_2$ or any I_0-centered P_3, and every C_4 and C_5 that contains v intersects I_0, then remove v from G.*

We call an annotated instance *reduced* if none of Rules 5.2–5.6 is applicable to this instance. The following lemma bounds the cardinalities of C_M and I_M in a reduced instance.

▶ **Lemma 11.** *If a reduced instance (G, I_0, k) is a yes-instance, then $|C_M| \leqslant 3k\sqrt{2k}$ and $|I_M| \leqslant k + 1$.*

Proof. Let E_- be any solution to (G, I_0, k) with at most k edges. Since Rule 5.6 is not applicable, every vertex in C_M is contained in some $2K_2$ or I_0-centered P_3, or some C_4 or C_5 in $G - I_0$. Any of these structures contains an edge in E_-. Therefore, to bound $|C_M|$, it suffices to count how many vertices in C_M can form a $2K_2$ or I_0-centered P_3, or a C_4 or C_5 in $G - I_0$ with an edge $xy \in E_-$.

- If a vertex $v \in C_M$ is in a $2K_2$ with edge xy, then either $v \in \{x, y\}$ or v is adjacent to neither x nor y. In the first case, no other vertex in C_M can occur in any $2K_2$ with xy. Since $xy \in E(G)$, at least one of them is not in I_0 (Rule 5.4). This vertex has at most $\sqrt{2k}$ non-neighbors in C_M. Therefore, the total number of vertices in C_M that can occur in any $2K_2$ with xy is at most $\sqrt{2k}$.
- If xy is an edge in any I_0-centered P_3, then precisely one of them is in I_0. Assume without loss of generality $x \in I_0$. If a vertex $v \in C_M$ is in an I_0-centered P_3 with the edge xy, then either $v = y$, or v is not adjacent to y. Since $y \notin I_0$, it has at most $\sqrt{2k}$ non-neighbors in C_M. Thus, the total number of vertices in C_M that can occur in any I_0-centered P_3 containing xy is at most $\sqrt{2k} + 1$.

- If a vertex $v \in C_M$ is in a C_4 or C_5 that contains xy, then v is adjacent to at most one of x and y. Since this C_4 or C_5 is in $G - I_0$, each of x and y has at most $\sqrt{2k}$ non-neighbors in C_M. Thus, the total number of vertices in C_M that can occur in such a C_4 or C_5 is at most $2\sqrt{2k}$.

Noting that an edge cannot satisfy the conditions of both the second ($|\{x, y\} \cap I_0| = 1$) and third ($|\{x, y\} \cap I_0| = 0$) categories, we can conclude $|C_M| \leqslant k(\sqrt{2k} + 2\sqrt{2k}) = 3k\sqrt{2k}$.

Since Rule 5.2 is not applicable, no vertex in I_M is simplicial. Suppose that $C \uplus I$ is a valid partition of G. Since C is a clique, for each vertex $v \in I_M \cap I$, at least one neighbor of v is in I. Therefore, each vertex $v \in I_M \cap I$ is incident to an edge in the solution $E(I, I)$. Noting that I_M is an independent set, we have $k \geqslant |I_M \cap I| \geqslant |I_M| - 1$, where the second inequality follows from Lemma 8(i). Thus, $|I_M| \leqslant k + 1$, and this concludes this proof. ◀

Note that the application of Rule 5.1 is different from the other ones. The application of one of Rules 5.2–5.6 may trigger the applicable of another. After the application of Rule 5.1, the instance is no longer annotated, and we will not go back to check the other rules. We summarize the algorithm in Algorithm 1.

Algorithm 1 A summary of our kernelization algorithm for split edge deletion.

INPUT: an instance (G, k) of the split edge deletion problem.
OUTPUT: an equivalent instance (G', k') with $|V(G')| = O(k'^{1.5})$.

1. $I_0 \leftarrow \emptyset$;
1. $M \leftarrow$ a maximal packing of vertex-disjoint $2K_2$'s, C_4's, and C_5's;
2. **if** $|M| > 5k$ **then return** a trivial no-instance;
3. **if** $k < 0$ **then return** a trivial no-instance;
4. **for each** simplicial vertex $v \in V(G) \setminus I_0$ **do** (Rule 5.2)
   if $|E(G - (N[v] \setminus I_0))| \leqslant k$ **then return** a trivial yes-instance;
   else $I_0 \leftarrow I_0 \cup \{v\}$;
5. remove c-vertices and mark i-vertices (Rule 5.3);
6. **if** $E(I_0, I_0) \neq \emptyset$ **then**
   remove edges in $E(I_0, I_0)$ and decrease k (Rule 5.4);
7. merge I_0 into $\leqslant k$ vertices (Rule 5.5);
8. remove vertices in C_M not contained in certain structures (Rule 5.6);
9. **if** any of Rules 5.2–5.4 and 5.6 made a change **then goto** 3;
10. **if** $|C_M| + |I_M| > 3k\sqrt{2k} + k + 1$ **then return** a trivial no-instance;
11. add $\sqrt{2k} + 1$ new vertices and remove all marks (Rule 5.1);
12. **return** (G, k).

▶ **Theorem 12.** *There is an $O(k^{1.5})$-vertex kernel for the split edge deletion problem.*

Proof. We use the algorithm described in Algorithm 1. The first two steps build the modulator, and their correctness follows from that any solution contains at least one edge of each forbidden induced subgraph of G. Step 3 is obviously correct. Steps 4–8 follow from the safeness of the rules; so is step 11. The correctness of step 10 is ensured by Lemma 11.

The cardinality of M is at most $5k$, and it never increases during the algorithm. After step 7, $|I_0| \leqslant k$. We have bounded the cardinalities of C_M and I_M in Lemma 11. Step 11 increases $|C_M|$ by $\sqrt{2k} + 1$. Putting them together, we have

$$|V(G)| \leqslant 5k + k + (3k\sqrt{2k} + \sqrt{2k} + 1) + k + 1 = O(k^{1.5}).$$

It is easy to verify that each reduction rule can be checked and applied in polynomial time. To see that the algorithm runs in polynomial time, note that if any of Rules 5.2–5.4 and 5.6 made a change to the instance, then either k decreases by one (Rule 5.4), or the cardinality of $V(G) \setminus I_0$ decreases by one (the other three rules). ◀

Since the class of split graphs is self-complementary, our algorithm also implies a kernel for the split completion problem. This kernel actually has fewer edges than the one for split edge deletion.

▶ **Theorem 13.** *There is a kernel of* $O(k^{1.5})$ *vertices and* $O(k^{2.5})$ *edges for the split completion problem.*

Proof. Let (G, k) be the input instance of the split completion problem. We can either take the complement of the input graph and consider it as an instance of the split edge deletion problem, or run the "complemented versions" of the rules. In the final result, we have an independent set of at most $O(k\sqrt{k})$ vertices, and at most $O(k)$ other vertices. The claim then follows. ◀

6 Pseudo-split edge deletion and pseudo-split completion

The algorithm in Alg. 1 also works for the pseudo-split ($\{2K_2, C_4\}$-free) edge deletion problem.

▶ **Theorem 14.** *There is an* $O(k^{1.5})$*-vertex kernel for the pseudo-split edge deletion problem. There is a kernel of* $O(k^{1.5})$ *vertices and* $O(k^{2.5})$ *edges for the pseudo-split completion problem.*

References

1. Jonathan F. Buss and Judy Goldsmith. Nondeterminism within P. *SIAM Journal on Computing*, 22(3):560–572, 1993. doi:10.1137/0222038.
2. 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.
3. Yixin Cao and Jianer Chen. Cluster editing: Kernelization based on edge cuts. *Algorithmica*, 64(1):152–169, 2012. doi:10.1007/s00453-011-9595-1.
4. Christophe Crespelle, Pål Grønås Drange, Fedor V. Fomin, and Petr A. Golovach. A survey of parameterized algorithms and the complexity of edge modification. arXiv:2001.06867, 2020.
5. Pål Grønås Drange, Markus Sortland Dregi, Daniel Lokshtanov, and Blair D. Sullivan. On the threshold of intractability. In Nikhil Bansal and Irene Finocchi, editors, *Proceedings of the 23rd*, volume 9294 of *LNCS*, pages 411–423. Springer, 2015. doi:10.1007/978-3-662-48350-3_35.
6. Pål Grønås Drange, Fedor V. Fomin, Michał Pilipczuk, and Yngve Villanger. Exploring the subexponential complexity of completion problems. *ACM Transactions on Computation Theory (TOCT)*, 7(4):1–38, 2015. doi:10.1145/2799640.
7. Pål Grønås Drange and Michal Pilipczuk. A polynomial kernel for trivially perfect editing. *Algorithmica*, 80(12):3481–3524, 2018. doi:10.1007/s00453-017-0401-6.
8. Maël Dumas, Anthony Perez, and Ioan Todinca. A Cubic Vertex-Kernel for Trivially Perfect Editing. In *46th International Symposium on Mathematical Foundations of Computer Science (MFCS 2021)*, volume 202 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 45:1–45:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.MFCS.2021.45.
9. Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006.
10. Esha Ghosh, Sudeshna Kolay, Mrinal Kumar, Pranabendu Misra, Fahad Panolan, Ashutosh Rai, and M. S. Ramanujan. Faster parameterized algorithms for deletion to split graphs. *Algorithmica*, 71(4):989–1006, 2015. doi:10.1007/s00453-013-9837-5.
11. Niels Grüttemeier and Christian Komusiewicz. On the relation of strong triadic closure and cluster deletion. *Algorithmica*, 82(4):853–880, 2020. doi:10.1007/s00453-019-00617-1.

12 Jiong Guo. Problem kernels for NP-complete edge deletion problems: Split and related graphs. In Takeshi Tokuyama, editor, *Proceedings of the 18th*, volume 4835 of *LNCS*, pages 915–926. Springer, 2007. doi:10.1007/978-3-540-77120-3_79.

13 Peter L. Hammer and Bruno Simeone. The splittance of a graph. *Combinatorica*, 1(3):275–284, 1981. doi:10.1007/BF02579333.

14 Athanasios L. Konstantinidis, Stavros D. Nikolopoulos, and Charis Papadopoulos. Strong triadic closure in cographs and graphs of low maximum degree. *Theoretical Computer Science*, 740:76–84, 2018. doi:10.1016/j.tcs.2018.05.012.

15 Federico Mancini. *Graph Modification Problems Related to Graph Classes*. PhD thesis, University of Bergen, Bergen, Norway, 2008.

16 Dániel Marx and R. B. Sandeep. Incompressibility of H-free edge modification problems: Towards a dichotomy. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *Proceedings of the 28th*, volume 173 of *LIPIcs*, pages 72:1–72:25. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.72.

17 Assaf Natanzon, Ron Shamir, and Roded Sharan. Complexity classification of some edge modification problems. *Discrete Applied Mathematics*, 113(1):109–128, 2001. doi:10.1016/S0166-218X(00)00391-7.

18 Roded Sharan. *Graph Modification Problems and their Applications to Genomic Research*. PhD thesis, Tel-Aviv University, Tel Aviv, Israel, 2002.

19 E. S. Wolk. The comparability graph of a tree. *Proceedings of the American Mathematical Society*, 13:789–795, 1962. doi:10.1090/S0002-9939-1962-0172273-0.

20 Jing-Ho Yan, Jer-Jeong Chen, and Gerard Jennhwa Chang. Quasi-threshold graphs. *Discrete Applied Mathematics*, 69(3):247–255, 1996. doi:10.1016/0166-218X(96)00094-7.

21 Mihalis Yannakakis. Edge-deletion problems. *SIAM Journal on Computing*, 10(2):297–309, 1981. doi:10.1137/0210021.

Preprocessing for Outerplanar Vertex Deletion: An Elementary Kernel of Quartic Size

Huib Donkers
Eindhoven University of Technology, The Netherlands

Bart M. P. Jansen
Eindhoven University of Technology, The Netherlands

Michał Włodarczyk
Eindhoven University of Technology, The Netherlands

Abstract

In the $\mathcal{F}$-Minor-Free Deletion problem one is given an undirected graph G, an integer k, and the task is to determine whether there exists a vertex set S of size at most k, so that $G - S$ contains no graph from the finite family $\mathcal{F}$ as a minor. It is known that whenever $\mathcal{F}$ contains at least one planar graph, then $\mathcal{F}$-Minor-Free Deletion admits a polynomial kernel, that is, there is a polynomial-time algorithm that outputs an equivalent instance of size $k^{\mathcal{O}(1)}$ [Fomin, Lokshtanov, Misra, Saurabh; FOCS 2012]. However, this result relies on non-constructive arguments based on well-quasi-ordering and does not provide a concrete bound on the kernel size.

We study the Outerplanar Deletion problem, in which we want to remove at most k vertices from a graph to make it outerplanar. This is a special case of $\mathcal{F}$-Minor-Free Deletion for the family $\mathcal{F} = \{K_4, K_{2,3}\}$. The class of outerplanar graphs is arguably the simplest class of graphs for which no explicit kernelization size bounds are known. By exploiting the combinatorial properties of outerplanar graphs we present elementary reduction rules decreasing the size of a graph. This yields a constructive kernel with $\mathcal{O}(k^4)$ vertices and edges. As a corollary, we derive that any minor-minimal obstruction to having an outerplanar deletion set of size k has $\mathcal{O}(k^4)$ vertices and edges.

2012 ACM Subject Classification Mathematics of computing → Graphs and surfaces; Mathematics of computing → Graph algorithms; Theory of computation → Graph algorithms analysis; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases fixed-parameter tractability, kernelization, outerplanar graphs

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.14

Related Version *Full Version*: https://arxiv.org/abs/2110.01868 [14]

Funding This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 803421, ReduceSearch).

1 Introduction

Background and Motivation. Kernelization [19] is a subfield of parameterized complexity [7, 15] that investigates the complexity of preprocessing NP-hard problems. A parameterized problem includes in its input an integer k which we call the parameter. This parameter can be seen as a measure of complexity of the problem input. A common choice is to treat the size of the desired solution as the parameter. A kernelization is a polynomial-time preprocessing algorithm that converts a problem instance with parameter k into an equivalent parameterized instance of the same problem such that both the size and the parameter value of the new instance are bounded by a function f of k. The function f is called the size of the kernel. It is known that a decidable parameterized problem has a kernel if and only if it is fixed-parameter tractable [7, Lemma 2.2]. A major challenge is to determine which parameterized problems admit a kernel of polynomial size.

One class of problems that received much attention [17, 18, 23, 26, 28] is $\mathcal{F}$-Minor-Free Deletion. For a fixed finite family of graphs $\mathcal{F}$, the $\mathcal{F}$-Minor-Free Deletion problem asks, given a graph G and parameter k, whether a vertex set $S \subseteq V(G)$ of size k exists such that the graph $G - S$, obtained from G by removing the vertices in S, does not contain any graph $F \in \mathcal{F}$ as a minor. This class of problems includes a large variety of well-studied problems such as Vertex Cover, Feedback Vertex Set, and Planarization, which are obtained by taking $\mathcal{F}$ equal to (respectively) $\{K_2\}$, $\{K_3\}$, and $\{K_5, K_{3,3}\}$. All of the $\mathcal{F}$-Minor-Free Deletion problems are fixed-parameter tractable [38], but it is unknown whether they all admit a polynomial kernel [18]. If each graph in $\mathcal{F}$ contains at least one edge, it follows from the general results of Lewis and Yannakakis [30] that $\mathcal{F}$-Minor-Free Deletion is NP-hard.

If $\mathcal{F}$ is restricted to only families containing a planar graph we speak of Planar-$\mathcal{F}$ Deletion. Since the family of $\mathcal{F}$-minor-free graphs has bounded treewidth if and only if $\mathcal{F}$ includes a planar graph [36], this restriction ensures that removing a solution to the problem yields a graph of constant treewidth. Hence any solution is a treewidth-η modulator for some $\eta \in \mathbb{N}$ depending on $\mathcal{F}$. For this more restricted class Fomin et al. [18] have shown that polynomial kernels exist for each choice of $\mathcal{F}$. However, the running time of this kernelization algorithm is described by the authors as "horrendous" and regarding the size the authors state the following:

> The size of the kernel, however, is not explicit. Several of the constants that go into the proof of Lemma 29 depend on the size of the largest graph in certain antichains in a well-quasi-order and thus we don't know what the (constant) exponent bounding the size of the kernel is. We leave it to future work to make also the size of the kernel explicit.

For some specific Planar-$\mathcal{F}$ Deletion problems kernels with explicit size are known. Most famous are Vertex Cover and Feedback Vertex Set which admit kernels with respectively a linear and quadratic number of vertices [5, 25, 42]. Additionally, if θ_c denotes the graph with two vertices and $c \geq 1$ parallel edges, then $\{\theta_c\}$-Minor-Free Deletion admits a kernel with $\mathcal{O}(k^2 \log^{3/2} k)$ vertices and edges [17, Theorem 1.2]; note that the cases $c = 1$ and $c = 2$ correspond to Vertex Cover and Feedback Vertex Set. Another problem for which an explicit kernel size bound is known is Pathwidth-one Deletion, where the goal is to obtain a graph of pathwidth one, i.e, each connected component is a caterpillar. First a kernel of quartic size was given [34] which was later improved to a quadratic kernel [8]. If we want to remove at most k vertices to reduce the treedepth to at most η, we obtain the Treedepth-η Deletion problem. Since this property can be characterized by forbidden minors and bounded treedepth implies bounded treewidth, this problem is also a special case of Planar-$\mathcal{F}$ Deletion. Giannopoulou et al. [23] have shown that for every η, there is a kernel with $2^{\mathcal{O}(\eta^2)} \cdot k^6$ vertices for Treedepth-η Deletion. They have also proven that in general there is no hope for a universal constant in the kernel exponent and the degree of the polynomial which bounds the kernel size must increase as a function of $\mathcal{F}$ unless $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$.

In this paper we investigate Outerplanar Deletion, which asks for a graph G and parameter k whether a set $S \subseteq V(G)$ of size k exists such that $G - S$ is outerplanar. A graph is outerplanar if it admits a planar embedding for which all vertices lie on the outer face, or equivalently, if it does not contain K_4 or $K_{2,3}$ as a minor. Outerplanar graphs form a rich superclass of forests and are frequently studied in graph theory [4, 6, 10, 16, 41], graph drawing [1, 21, 32], and optimization [22, 31, 33, 35].

Since outerplanarity can be characterized as being $\{K_4, K_{2,3}\}$-minor-free [4], the problem belongs to the class of Planar-$\mathcal{F}$ Deletion problems. It is arguably the easiest problem in the class for which no explicit polynomial kernel is known. This makes Outerplanar Deletion a well-suited starting point to deepen our understanding of Planar-$\mathcal{F}$ Deletion problems in the search for explicit kernelization bounds.

Results. Let $\mathsf{opd}(G)$ denote the minimum size of a vertex set $S \subseteq V(G)$ such that $G - S$ is outerplanar. Our main result is the following theorem:

▶ **Theorem 1.1.** *The Outerplanar Deletion problem admits a polynomial-time kernelization algorithm that, given an instance (G, k), outputs an equivalent instance (G', k'), such that $k' \leq k$, graph G' is a minor of G, and G' has $\mathcal{O}(k^4)$ vertices and edges. Furthermore, if $\mathsf{opd}(G) \leq k$, then $\mathsf{opd}(G') = \mathsf{opd}(G) - (k - k')$.*

The algorithm behind Theorem 1.1 is elementary, consisting of a subroutine to build a decomposition of the input graph G using marking procedures in a tree decomposition, together with a series of explicit reduction rules. In particular, we avoid the use of protrusion replacement (summarized below). Concrete bounds on the hidden constant in the $\mathcal{O}$-notation follow from our arguments. The size bound depends on the approximation ratio of an approximation algorithm that bootstraps the decomposition phase, for which the current state-of-the-art is 40. We will therefore present a formula to obtain a concrete bound on the kernel size, rather than its value using the current-best approximation (which would exceed 10^5).

Theorem 1.1 presents the first concrete upper bound on the degree of the polynomial that bounds the size of kernels for Outerplanar Deletion. We hope that it will pave the way towards obtaining explicit size bounds for all Planar-$\mathcal{F}$ Deletion problems and give an impetus for research on the kernelization complexity of the Planar Deletion problem, which is one of the major open problems in kernelization today [40, 4:28],[19, Appendix A].

Via known connections [18] between kernelizations that reduce to a minor of the input graph and bounds on the sizes of obstruction sets, we obtain the following corollary.

▶ **Corollary 1.2.** *If G is a graph such that $\mathsf{opd}(G) > k$ but each proper minor G' of G satisfies $\mathsf{opd}(G') \leq k$, then G has $\mathcal{O}(k^4)$ vertices and edges.*

The existence of a polynomial bound with unknown degree follows from the work of Fomin et al. [18]; Corollary 1.2 gives the first explicit size bounds and contributes to a large body of research on minor-order obstructions (e.g. [3, 11, 12, 13, 29, 37, 39]).

Techniques. The known kernelization algorithms [17, 18] for Planar-$\mathcal{F}$ Deletion make use of (near-)protrusions. A protrusion is a vertex set that induces a subgraph of constant treewidth and boundary size. Protrusion replacement is a technique where sufficiently large protrusions are replaced by smaller ones without changing the answer. Protrusion techniques were first used to obtain kernels for problems on planar and other topologically-defined graph classes [2]. Later Fomin at al. [17] described how to use protrusion techniques for problems on general graphs. They proved [17, Lemma 3.3] that any graph G, which contains a modulator X to constant treewidth such that $|X|$ and the size of its neighborhood can be bounded by a polynomial in k, contains a protrusion of size $|V(G)|/k^{\mathcal{O}(1)}$ that can be found efficiently. For any fixed $\mathcal{F}$ containing a planar graph, they present a method to obtain a small modulator to an $\mathcal{F}$-minor-free graph, which has constant treewidth. This leads to a polynomial kernel for Planar-$\mathcal{F}$ Deletion on graphs with bounded degree since the

size of the neighborhood of the modulator can be bounded so protrusion replacement can be used to obtain a polynomial kernel. Specifically for $\{\theta_c\}$-Minor-Free Deletion they give reduction rules to reduce the maximum degree in a general graph, which leads to a polynomial kernel on general graphs.

The kernel for Planar-$\mathcal{F}$ Deletion given by Fomin et al. [18] does not rely on bounding the size of the neighborhood of the modulator followed by protrusion replacement. Instead they present the notion of a near-protrusion: a vertex set that will become a protrusion after removing any size-k solution from the graph. With an argument based on well-quasi-ordering they determine that if such near-protrusions are large enough one can, in polynomial time, reduce to a proper minor of the graph without changing the answer.

In this paper we present a method for Outerplanar Deletion to decrease the size of the neighborhood of a modulator to outerplanarity. This relies on a process that was called "tidying the modulator" in earlier work [43] and also used in the kernelization for Chordal Vertex Deletion [27]. The result is a larger modulator $X \subseteq V(G)$ but with the additional feature that it retains its modulator properties when omitting any single vertex, that is, $G - (X \setminus \{x\})$ is outerplanar for each $x \in X$. We proceed by decomposing the graph into near-protrusions, following along similar lines as the decomposition by Fomin et al. [17] but exploiting the structure of outerplanar graphs at several steps to obtain such a decomposition with respect to our larger tidied modulator, without leading to worse bounds. With the additional properties of the modulator X obtained from tidying we no longer need to rely on well-quasi-ordering, but instead are able to reduce the size of the neighborhood of the modulator in two steps. The first reduces the number of connected components of $G - X$ which are adjacent to any particular modulator vertex $x \in X$. In the case of $\{\theta_c\}$-minor-free graphs, if $G - (X \setminus \{x\})$ is $\{\theta_c\}$-minor-free then bounding the number of components of $G - X$ adjacent to each $x \in X$ this is sufficient to bound $|N_G(X)|$, since any $x \in X$ has less than c neighbors in any component of $G - (X \setminus \{x\})$. One of the major difficulties we face when working with $\{K_{2,3}\}$-minor-free graphs is that in such a graph there can be arbitrarily many edges between a vertex x and a connected component of $G - (X \setminus \{x\})$. Therefore we present an additional reduction rule that reduces, in a second step, the number of edges between a vertex and a connected component. After these two steps we obtain a bound on the size of the neighborhood of the modulator. At this point, standard protrusion replacement could be applied to prove the *existence* of a kernel for Outerplanar Deletion with $\mathcal{O}(k^4)$ vertices. In order to give an explicit kernelization algorithm we present a number of additional reduction rules to avoid the generic protrusion replacement technique. This eventually leads to a kernel with at most $c \cdot k^4$ vertices and edges for Outerplanar Deletion. It is conceptually simple (yet tedious) to extract the explicit value of c from the algorithm description.

Organization. In the next section we give basic definitions and notation we use throughout the rest of the paper, together with structural observations for outerplanar graphs. Section 3 describes how we obtain small modulators to outerplanarity with progressively stronger properties, and finally we obtain a modulator of size $\mathcal{O}(k^4)$ such that each remaining component has only 4 neighbors in the modulator, effectively forming a decomposition into protrusions. The second stage of the kernelization reduces the size of the connected components outside the modulator. These reduction rules are described in Section 4. In Section 5 we finally tie everything together to obtain a kernel with $\mathcal{O}(k^4)$ vertices and edges. Due to space restrictions, all proofs (except Lemma 3.17) have been deferred to the full version [14].

2 Preliminaries

Kernelization. A parameterized problem is a decision problem in which every input has an associated positive integer parameter that captures its complexity in some well-defined way. For a parameterized problem $A \subseteq \Sigma^* \times \mathbb{N}$ and a function $f\colon \mathbb{N} \to \mathbb{N}$, a kernelization for A of size f is an algorithm that, on input $(x,k) \in \Sigma^* \times \mathbb{N}$, takes time polynomial in $|x|+k$ and outputs $(x',k') \in \Sigma^* \times \mathbb{N}$ such that the following holds:

1. $(x,k) \in A$ if and only if $(x',k') \in A$, and
2. both $|x'|$ and k' are bounded by $f(k)$.

Graph theory. The set $\{1,\ldots,p\}$ is denoted by $[p]$. We consider simple undirected graphs without self-loops. A graph G has vertex set $V(G)$ and edge set $E(G)$. We use shorthand $n = |V(G)|$ and $m = |E(G)|$. For (not necessarily disjoint) $A, B \subseteq V(G)$, we define $E_G(A,B) = \{uv \mid u \in A, v \in B, uv \in E(G)\}$. The open neighborhood of $v \in V(G)$ is $N_G(v) := \{u \mid uv \in E(G)\}$, where we omit the subscript G if it is clear from context. For a vertex set $S \subseteq V(G)$ the open neighborhood of S, denoted $N_G(S)$, is defined as $\bigcup_{v\in S} N_G(v) \setminus S$. The closed neighborhood of a single vertex v is $N_G[v] := N_G(v) \cup \{v\}$, and the closed neighborhood of a vertex set S is $N_G[S] := N_G(S) \cup S$. The boundary of a vertex set $S \subseteq V(G)$ is the set $\partial_G(S) = N_G(V(G) \setminus S)$. For $A \subseteq V(G)$, the graph induced by A is denoted by $G[A]$ and we say that the vertex set A is connected if the graph $G[A]$ is connected. We use notation $G\langle A\rangle = G[N_G[A]]$ and, when H is an induced subgraph of G, we write briefly $G\langle H\rangle = G\langle V(H)\rangle$ or $\partial_G(H) = \partial_G(V(H))$. We use shorthand $G - A$ for the graph $G[V(G) \setminus A]$. For $v \in V(G)$, we write $G - v$ instead of $G - \{v\}$. For $A \subseteq E(G)$ we denote by $G \setminus A$ the graph with vertex set $V(G)$ and edge set $E(G) \setminus A$. For $e \in E(G)$ we write $G \setminus e$ instead of $G \setminus \{e\}$. If $e = uv$, then $V(e) = \{u,v\}$.

A vertex $v \in V(G)$ is an articulation point in a connected graph G if $G - v$ is not connected. A graph is called biconnected if it has no articulation points. A biconnected component in G is an inclusion-wise maximal subgraph which is biconnected. A vertex set $A \subseteq V(G)$ is an independent set in G if $E_G(A,A) = \emptyset$. A graph G is bipartite if there is a partition of $V(G)$ into two independent sets A, B. We write shortly $G = (A \cup B, E)$ to specify a bipartite graph on vertex set $E = E(G)$ admitting this partition.

▶ **Definition 2.1.** *For a vertex set $X \subseteq V(G)$ the* component graph $\mathcal{C}(G,X)$ *is a bipartite graph $(X \cup Y, E)$, where Y is the set of connected components of $G - X$, and $(v,C) \in E$ if there is at least one edge between $v \in X$ and the component $C \in Y$.*

For an integer q, the graph K_q is the complete graph on q vertices. For integers p, q, the graph $K_{p,q}$ is the bipartite graph $(A \cup B, E)$, where $|A| = p$, $|B| = q$, and $uv \in E$ whenever $u \in A, v \in B$.

Minors. A contraction of $uv \in E(G)$ introduces a new vertex adjacent to all of $N_G(\{u,v\})$, after which u and v are deleted. The result of contracting $uv \in E(G)$ is denoted G/uv. For $A \subseteq V(G)$ such that $G[A]$ is connected, we say we contract A if we simultaneously contract all edges in $G[A]$ and introduce a single new vertex. We say that H is a minor of G, if we can turn G into H by a (possibly empty) series of edge contractions, edge deletions, and vertex deletions. If this series is non-empty, then H is called a proper minor of G.

Planar and outerplanar graphs. A graph is called planar if it admits a plane embedding. By Wagner's theorem, a graph G is planar if and only if G contains neither K_5 nor $K_{3,3}$ as a minor. A graph is called outerplanar if it admits a plane embedding with all vertices lying on

the outer face. A graph G is outerplanar if and only if G contains neither K_4 nor $K_{2,3}$ as a minor [4]. A graph G is planar (resp. outerplanar) if and only if every biconnected component in G induces a planar (resp. outerplanar) graph. Recall that $G\langle C\rangle = G[N_G(V(C))]$.

▶ **Observation 2.2.** *Let $v \in V(G)$. The graph G is outerplanar if and only if for each connected component C of $G - v$ the graph $G\langle C\rangle$ is outerplanar.*

For a graph G we call $S \subseteq V(G)$ an outerplanar deletion set if $G - S$ is outerplanar. The outerplanar deletion number of G, denoted $\mathsf{opd}(G)$, is the size of a smallest outerplanar deletion set in G.

Structural properties of outerplanar graphs. We present a number of structural observations of outerplanar graphs which will be useful in our later argumentation. The first is a characterization of outerplanar graphs similar to Observation 2.2. Rather than looking at the components of a graph with one vertex removed, it considers the components of a graph with both endpoints of an edge removed. This allows us for example to easily argue about outerplanarity of graphs obtained from "gluing" two outerplanar graphs on two adjacent vertices.

▶ **Lemma 2.3.** *Let G be a graph and $e \in E(G)$. Then G is outerplanar if and only if both of the following conditions hold:*

1. *for each connected component C of $G - V(e)$ the graph $G\langle C\rangle$ is outerplanar, and*
2. *the graph $G \setminus e$ does not have three induced internally vertex-disjoint paths connecting the endpoints of e.*

In order to more easily apply Lemma 2.3, we show that no two induced paths as referred to in Lemma 2.3(2) can lie in the same connected component C as referred to in Lemma 2.3(1).

▶ **Lemma 2.4.** *Suppose G is outerplanar with an edge $uv \in E(G)$. If P_1, P_2 are internally vertex-disjoint (u, v)-paths in $G \setminus uv$, then the interiors of P_1 and P_2 lie in different connected components of $G - \{u, v\}$.*

We now give a condition under which an edge can be added to an outerplanar without violating outerplanarity. Intuitively, this corresponds to adding an edge between two vertices that lie on the same interior face.

▶ **Lemma 2.5.** *Suppose G is outerplanar and vertices x, y lie on an induced cycle D with $xy \notin E(G)$. Then adding the edge xy to G preserves outerplanarity.*

Finally, we observe that if an outerplanar graph G has a cycle C, then any component of $G - V(C)$ is adjacent to at most two vertices of the cycle (else there would be a K_4 minor), and these must be consecutive on the cycle (else there would be a $K_{2,3}$ minor).

▶ **Lemma 2.6.** *If C is a cycle in an outerplanar graph G, then each connected component of $G - V(C)$ has at most two neighbors in C, and they must be consecutive along the cycle.*

3 Splitting the graph into pieces

In this section we show how to reduce any input of Outerplanar Deletion to an equivalent instance which admits a decomposition into a modulator of bounded size along with a bounded number of outerplanar components containing at most four neighbors of the modulator.

3.1 The augmented modulator

The starting point for both our kernelization algorithm and the one from Fomin et al. [18] is to employ a constant-factor approximation algorithm. We however begin with a different approximation algorithm, which has two advantages. First, the algorithm is constructive: it relies only on separating properties of bounded-treewidth graphs and rounding a fractional solution from a linear programming relaxation. Second, the approximation factor can be pinned down to a concrete value. In the full version, we show how the general theorem by Gupta et al. [24] implies the following.

▶ **Theorem 3.1** ([24]). *There is a polynomial-time deterministic 40-approximation algorithm for* Outerplanar Deletion.

In our setting, for a given graph G and integer k, we want to determine whether G admits an outerplanar deletion set of size at most k. Thanks to the theorem above, we can assume that we are given an outerplanar deletion set X (also called a modulator to outerplanarity) of size at most $40 \cdot k$. As a next step, we would like to augment this set to satisfy a stronger property. This step is inspired by the technique of tidying the modulator from van Bevern, Moser, and Niedermeier [43]. For each vertex $v \in X$ we would like to be able to "put it back" into $G - X$ while maintaining outerplanarity. In order to do so, we look for a set of vertices from $V(G) \setminus X$ that needs to be removed if v is put back. Since $G - X$ is outerplanar and hence has treewidth at most two, we can construct such a set of moderate size by a greedy approach. We scan a tree decomposition in a bottom-up manner and look for maximal subgraphs that are outerplanar when considered together with v. When such a subgraph cannot be further extended we mark one bag of a decomposition, which gives 3 vertices to be removed. We show that this idea leads to a 3-approximation algorithm. While this approach based on covering/packing duality is well-known, for completeness we include the proof in the full version.

▶ **Lemma 3.2.** *There is a polynomial-time algorithm that, given a graph G, an integer k, and a vertex v such that $G - v$ is outerplanar, either finds an outerplanar deletion set $S \subseteq V(G) \setminus \{v\}$ in G of size of most $3k$ or correctly concludes that there is no outerplanar deletion set $S \subseteq V(G) \setminus \{v\}$ in G of size of most k.*

Observe that if it is impossible to remove k vertices from $G - (X \setminus \{v\})$ to make it outerplanar, then any outerplanar deletion set in G of size at most k must contain v. In this situation it suffices to solve the problem on $G - v$. Otherwise, we identify a set $R(v)$ of at most $3k$ vertices whose removal allows v to be put back in $G - X$ without spoiling outerplanarity. After inserting $R(v)$ into the set X, we could put v back "for free". Let us formalize this idea of augmenting the modulator.

▶ **Definition 3.3.** *A* (k, c)-augmented modulator *in graph G is a pair of disjoint sets $X_0, X_1 \subseteq V(G)$ such that:*

1. *$G - X_0$ is outerplanar,*
2. *for each $v \in X_0$, there is a set $R(v) \subseteq X_1$, such that $|R(v)| \leq 3k$ and $G - ((X_0 \setminus \{v\}) \cup R(v))$ is outerplanar, and*
3. *$|X_0| \leq c \cdot k$, $X_1 = \bigcup_{v \in X_0} R(v)$, which implies $|X_1| \leq 3c \cdot k^2$.*

We classify the pairs of vertices within $X_0 \cup X_1$. A pair (u, v): $u, v \in X_0 \cup X_1$ is of type:
A: if $u, v \in X_0$ or $(u \in X_0, v \in R(u))$ or $(v \in X_0, u \in R(v))$,
B: if (u, v) is not of type A and $\{u, v\} \cap X_0 \neq \emptyset$,
C: if $u, v \in X_1$.
We note that the number of type-A pairs is at most $c(3 + c) \cdot k^2$, the number of type-B pairs is at most $3c^2 \cdot k^3$, and the number of type-C pairs is at most $9c^2 \cdot k^4$.

The downside of the augmented modulator is that its size can be as large as $\mathcal{O}(k^2)$. However, in return we obtain an even stronger property than previously sketched. For most of the pairs of vertices u, v from the augmented modulator (X_0, X_1), putting them back into $G - (X_0 \cup X_1)$ at the same time still does not break outerplanarity. This property will come in useful for bounding the size of the kernel.

▶ **Observation 3.4.** *Let (X_0, X_1) be a (k, c)-augmented modulator in a graph G. Then for each $v \in X_0 \cup X_1$, the graph $G - (X_0 \cup X_1 \setminus \{v\})$ is outerplanar. Furthermore, if $u, v \in X_0 \cup X_1$ and the pair (u, v) is of type B or C, then the graph $G - (X_0 \cup X_1 \setminus \{u, v\})$ is outerplanar.*

Let us summarize what we can compute so far. We say that instances (G, k) and (G', k') are equivalent if $\mathsf{opd}(G) \leq k \Leftrightarrow \mathsf{opd}(G') \leq k'$.

▶ **Lemma 3.5.** *There is a polynomial-time algorithm that, given an instance (G, k), either correctly concludes that $\mathsf{opd}(G) > k$ or outputs an equivalent instance (G', k'), where $k' \leq k$ and G' is a subgraph of G, along with a $(k', 40)$-augmented modulator in G'. If $\mathsf{opd}(G) \leq k$ then it holds that $\mathsf{opd}(G') = \mathsf{opd}(G) - (k - k')$. Moreover, if for every vertex $v \in V(G)$ there is an outerplanar deletion set $S \subseteq V(G) \setminus \{v\}$ in G of size at most k, then $k' = k$.*

The reduction step above is the only one in our algorithm that may decrease the value of k. Moreover, no further reduction will modify the outerplanar deletion number as long as $\mathsf{opd}(G) \leq k$. This observation will come in useful for bounding the size of minimal minor obstructions to having an outerplanar deletion set of size k.

As the next step, we would like to bound the number of connected components in $G - (X_0 \cup X_1)$ and the number of connections between the components and the modulator vertices. We show that if vertices $u, v \in X_0 \cup X_1$ are adjacent to sufficiently many components, then at least one of u, v must be removed in any solution of size at most k. Together with the "putting back" property of the augmented modulator, this allows us to forget some of the edges without modifying the space of solutions of size at most k. We formalize this idea with the following marking scheme.

▶ **Reduction Rule 1.** *Let G be a graph, $k \in \mathbb{N}$, and (X_0, X_1) be a (k, c)-augmented modulator in G. Consider the component graph $\mathcal{C}(G, X_0 \cup X_1)$. For each pair $u, v \in X_0 \cup X_1$ choose up to $k + 3$ components C_i with edges to both u and v, and mark the edges $(u, C_i), (v, C_i)$ in $\mathcal{C}(G, X_0 \cup X_1)$. If an edge (v, C) is unmarked in the end, remove all the edges between v and C in G. If some component C of $G - (X_0 \cup X_1)$ or a vertex $v \in X_0 \cup X_1$ becomes isolated, remove it from G.*

▶ **Lemma 3.6** (Safeness)**.** *Let G be a graph, $k \in \mathbb{N}$, and (X_0, X_1) be a (k, c)-augmented modulator in G. Let G' be obtained from G by applying Reduction Rule 1 with respect to (X_0, X_1, k). If $\mathsf{opd}(G) > k$ then $\mathsf{opd}(G') > k$ and if $\mathsf{opd}(G) \leq k$ then $\mathsf{opd}(G') = \mathsf{opd}(G)$.*

We show that after application of Reduction Rule 1 the component graph $\mathcal{C}(G, X_0 \cup X_1)$ cannot be too large. This will come in useful for proving further upper bounds. We could trivially bound the number of its edges by $|X_0 \cup X_1|^2 \cdot (k + 3) = \mathcal{O}(k^5)$ but, thanks to the properties of the augmented modulator, we can be more economical.

Recall the types of pairs from Definition 3.3 and their properties from Observation 3.4. We know that the number of type-A pairs is at most $c(3 + c) \cdot k^2$ and the number of type-B pairs is at most $3c^2 \cdot k^3$. The pairs of type B can be inserted back into $G - (X_0 \cup X_1)$ without affecting its outerplanarity. This implies that each type-B pair is responsible for marking at most 2 edges. Finally, the total number of edges marked due to type-C pairs is $\mathcal{O}(k^2)$.

▶ **Lemma 3.7.** *After the application of Rule 1 with respect to a (k,c)-augmented modulator (X_0, X_1), the number of vertices and edges in $\mathcal{C}(G, X_0 \cup X_1)$ is at most $f_1(c) \cdot (k+3)^3$, where $f_1(c) = 14c^2 + 60c$.*

3.2 The outerplanar decomposition

We proceed by enriching the augmented modulator further. We would like to provide additional properties at the expense of growing the modulator size to $\mathcal{O}(k^3)$. For two vertices u, v in an augmented modulator (X_0, X_1) ideally we would like to ensure that no two components of $G - (X_0 \cup X_1 \cup Z)$ are adjacent to both u and v, where Z is some vertex set of size $\mathcal{O}(k^3)$. This is not always possible, but we will guarantee that in such a case any outerplanar deletion set of size at most k must contain either u or v.

▶ **Definition 3.8.** *Let $Y \subseteq V(G)$ be a vertex subset in a graph G. We say that $u, v \in Y$ are Y-separated if no connected component of $G - Y$ is adjacent to both u and v.*

In Lemma 3.9 we are going to show that when G is outerplanar and $X \subseteq V(G)$, then there always exists a small set $Y \subseteq V(G)$ so that every pair from X is $(X \cup Y)$-separated.

▶ **Lemma 3.9.** *There is a polynomial-time algorithm that, given a vertex set $X \subseteq V(G)$ in an outerplanar graph G, finds a vertex set $Y \subseteq V(G) \setminus X$ of size at most $4 \cdot |X|$, so that every pair $u, v \in X$ with $u \neq v$ is $(X \cup Y)$-separated.*

Given an augmented modulator (X_0, X_1), we would like to find a set Z of moderate size so that for each pair (u, v) from $X_0 \cup X_1$ either u, v are $(X_0 \cup X_1 \cup Z)$-separated or there exist $k+4$ internally vertex-disjoint paths, with non-empty interior, connecting u and v in G. If the latter case occurs, then any outerplanar deletion set of size bounded by k, can intersect at most k of these paths' interiors. Therefore, this solution must remove either u or v in order to get rid of all $K_{2,3}$-minors. We remark that this property already holds if we request $k+3$ disjoint (u, v)-paths, but in this stronger form it also holds for a graph obtained from G by an edge removal. This fact will be crucial for the safeness proof for Reduction Rule 3.

In order to find the set Z, we could consider all pairs (u, v) from $X_0 \cup X_1$ and, if there exists an (u, v)-separator of size at most $k+3$, add it to Z. This however would make Z as large as $\mathcal{O}(k^5)$. We can make this process more economical by analyzing what happens for different types of pairs from Definition 3.3, similarly as in Lemma 3.7.

▶ **Lemma 3.10.** *There is a polynomial-time algorithm that, given an instance (G, k) with (k, c)-augmented modulator (X_0, X_1), returns a set $Z \subseteq V(G) \setminus (X_0 \cup X_1)$ of size at most $f_2(c) \cdot (k+3)^3$, where $f_2(c) = 4c^2 + 15c$, such that for each pair $u, v \in X_0 \cup X_1$ of distinct vertices one of the following holds:*

1. *vertices u, v are $(X_0 \cup X_1 \cup Z)$-separated, or*
2. *there are $k+4$ vertex-disjoint paths, with non-empty interior, connecting u and v in G.*

We would like to simplify the interface between a connected component C of $G - (X_0 \cup X_1 \cup Z)$ and the rest of the graph. Since $G - X_0$ is outerplanar, it has treewidth at most two, which implies there is a tree decomposition in which each pair of distinct bags intersects in at most 2 vertices. When constructing a separator $Z' \supseteq Z$ via the lowest common ancestor closure (see [20, §9.3.3]), the neighborhood of each connected component C of $G - Z'$ within the set Z' is contained in at most two bags of the decomposition. This allows us to guarantee that $|N_G(C) \cap Z'| \leq 4$.

▶ **Lemma 3.11.** *There is a polynomial-time algorithm that, given an outerplanar graph G and $Z \subseteq V(G)$, returns a set $Z' \supseteq Z$ of size at most $6 \cdot |Z|$ such that each connected component of $G - Z'$ has at most four neighbors in Z'.*

In order to keep the kernel size in check, we need to analyze the number of connected components of $G - (X_0 \cup X_1 \cup Z)$. We have managed to bound the size of Z by $\mathcal{O}(k^3)$ and, in Lemma 3.7, we have also bounded by $\mathcal{O}(k^3)$ the number of edges in the component graph $\mathcal{C}(G, X_0 \cup X_1)$. These two properties suffice to also bound the number of connected components of $G - (X_0 \cup X_1 \cup Z)$ that have at least two neighbors in $X_0 \cup X_1 \cup Z$. It will be easier to deal with the remaining ones later.

▶ **Lemma 3.12.** *Let (X_0, X_1) be a (k, c)-augmented modulator in G, so that the component graph $\mathcal{C}(G, X_0 \cup X_1)$ has at most s vertices and s edges, and let $Z \subseteq V(G) \setminus (X_0 \cup X_1)$. Then there are at most $3 \cdot s + 4 \cdot |Z|$ components of $G - (X_0 \cup X_1 \cup Z)$ that have two or more neighbors in $X_0 \cup X_1 \cup Z$.*

The previous lemma gives us a bound on the number of components outside the modulator with at least two neighbors. To bound the total number of components outside the modulator, we employ the following reduction rule to remove the remaining components with at most one neighbor.

▶ **Reduction Rule 2.** *If for some $C \subseteq V(G)$ the graph $G\langle C \rangle$ is outerplanar and it holds that $|N_G(C)| \leq 1$, then remove the vertex set C.*

Safeness of this rule follows from Observation 2.2, which implies $\mathsf{opd}(G - C) = \mathsf{opd}(G)$.

With these properties at hand, we are able to construct the desired extension of the augmented modulator. The decomposition below is inspired by the notion of a near-protrusion [18], combined with the idea of the augmented modulator, and with an $\mathcal{O}(k^3)$ bound on the number of leftover connected components.

▶ **Definition 3.13.** *For $k, c, d \in \mathbb{N}$ a (k, c, d)-outerplanar decomposition of a graph G is a triple (X_0, X_1, Z) of disjoint vertex sets in G, such that:*

1. *(X_0, X_1) is a (k, c)-augmented modulator for (G, k),*
2. *for each pair $u, v \in X_0 \cup X_1$ of distinct vertices one of the following holds:*
   a. *vertices u, v are $(X_0 \cup X_1 \cup Z)$-separated, or*
   b. *there are $k + 4$ vertex-disjoint (u, v)-paths in G, each with non-empty interior.*
3. *for each connected component C of $G - (X_0 \cup X_1 \cup Z)$ it holds that $|N_G(C) \cap Z| \leq 4$,*
4. *$|Z| \leq d \cdot (k+3)^3$ and there are at most $d \cdot (k+3)^3$ connected components in $G - (X_0 \cup X_1 \cup Z)$.*

▶ **Lemma 3.14.** *There is a constant c, a function $f_3 \colon \mathbb{N} \to \mathbb{N}$, and a polynomial-time algorithm that, given an instance (G, k), either returns an equivalent instance (G', k'), where $k' \leq k$ and G' is subgraph of G, along with a $(k', c, f_3(c))$-outerplanar decomposition of G', or concludes that $\mathsf{opd}(G) > k$. If $\mathsf{opd}(G) \leq k$ then it holds that $\mathsf{opd}(G') = \mathsf{opd}(G) - (k - k')$. Furthermore, $c = 40$ and $f_3(c) = 3 \cdot f_1(c) + 24 \cdot f_2(c)$ (see Lemmas 3.7 and 3.10).*

As the last property of the (k, c, d)-outerplanar decomposition, we formulate the bound on the total number of connections between $X_0 \cup X_1 \cup Z$ and the leftover components, which will lead to the total kernel size $\mathcal{O}(k^4)$.

▶ **Lemma 3.15.** *Let (X_0, X_1, Z) be a (k, c, d)-outerplanar decomposition of a graph G. Then the number of edges in the component graph $\mathcal{C}(G, X_0 \cup X_1 \cup Z)$ is at most $f_4(c, d) \cdot (k + 3)^4$, where $f_4(c, d) = cd + 6c + 4d$.*

3.3 Reducing the size of the neighborhood

Given a (k, c, d)-outerplanar decomposition (X_0, X_1, Z), we will now present the final reduction rule to reduce the size of the neighborhood $N_G(X_0 \cup X_1)$ to $\mathcal{O}(k^4)$. As the size of Z is already bounded by $\mathcal{O}(k^3)$ we focus on reducing the size of $N_G(X_0 \cup X_1) \setminus Z$. We have already shown the number of edges in the component graph $\mathcal{C}(G, X_0 \cup X_1 \cup Z)$ is bounded by $\mathcal{O}(k^4)$, so it suffices to reduce the number of edges between a single modulator vertex $x \in X_0 \cup X_1$ and a connected component C of $G - (X_0 \cup X_1 \cup Z)$ to a constant. For this, we first show in Lemma 3.16 where the neighbors of x occur in C.

In the following lemma, we consider an outerplanar graph G containing a vertex x. When omitting vertex x from a drawing of G, the vertices of $N_G(x)$ remain on the outer face of the graph. If $G - x$ is still connected, then there is a subpath P of the outer face which visits all of $N_G(x)$. The outerplanarity of G ensures that P can be chosen to be induced and to contain at most two vertices from each biconnected component of $G - x$. Furthermore, it follows from Lemma 2.6 that the latter must be consecutive. This is formalized as follows.

▶ **Lemma 3.16.** *Suppose G is outerplanar, $x \in V(G)$, and $G - x$ is connected. Then the vertices from $N_G(x)$ lie on an induced path P in $G - x$ such that for each biconnected component B of $G - x$ and each pair of distinct vertices $u, v \in V(P) \cap V(B)$ we have that $uv \in E(G - x)$. We can find such a path in polynomial time.*

We now investigate what happens when a modulator vertex $x \in X_0 \cup X_1$ is the only vertex in $X_0 \cup X_1$ that is adjacent to a connected component C of $G - (X_0 \cup X_1 \cup Z)$. If x has sufficiently many edges to a part of C that is not adjacent to Z, then one of these edges can be removed without affecting the outerplanar deletion number $\mathsf{opd}(G)$. We will also exploit this property for a reduction rule later in this paper when we reduce the number of edges within a connected component of $G - (X_0 \cup X_1 \cup Z)$ (see Figure 3, left side). Since the following lemma is the key ingredient in our algorithm, we include its full proof.

▶ **Lemma 3.17.** *Suppose we are given a graph G, a vertex $x \in V(G)$, and five vertices $v_1, \ldots, v_5 \in N_G(x)$ that lie, in order of increasing index, on an induced path P in $G - x$ from v_1 to v_5, such that $N_G(x) \cap V(P) = \{v_1, \ldots, v_5\}$. Let C be the component of $G - \{v_1, v_5, x\}$ containing $P - \{v_1, v_5\}$. If $G\langle C\rangle$ is outerplanar, then $\mathsf{opd}(G) = \mathsf{opd}(G \setminus xv_3)$.*

Proof. Clearly for any $S \subseteq V(G)$ if $G - S$ is outerplanar, then $G \setminus xv_3 - S$ is also outerplanar, hence $\mathsf{opd}(G) \geq \mathsf{opd}(G \setminus xv_3)$. To show $\mathsf{opd}(G) \leq \mathsf{opd}(G \setminus xv_3)$, suppose $G \setminus xv_3 - S$ is outerplanar for some arbitrary $S \subseteq V(G)$. If $x \in S$ or $v_3 \in S$ then clearly $G - S$ is outerplanar, so suppose $x, v_3 \notin S$. We show $G - S'$ is outerplanar for some $S' \subseteq V(G)$ with $|S'| \leq |S|$. Consider the following cases:

1. If $|S \cap V(P)| = 0$ then $G \setminus xv_3 - S$ contains an induced cycle formed by x together with the subpath of P from v_2 to v_4. This cycle includes x and v_3, so by Lemma 2.5 the graph $G \setminus xv_3 - S$ remains outerplanar after adding the edge xv_3, hence $G - S$ is outerplanar.
2. If $|S \cap V(P)| \geq 2$ then let $S' := \{v_1, v_5\} \cup (S \setminus V(C))$. Since $|S'| \leq |S|$, showing that $G - S'$ is outerplanar proves the claim. Let $\overline{C} := G - V(C)$ and note that $\overline{C} - S'$ is outerplanar since it is a subgraph of $G \setminus xv_3 - S$. Also note that $G[V(C) \cup \{x\}]$ is outerplanar since it is a subgraph of $G[V(C) \cup \{v_1, v_5, x\}] = G\langle C\rangle$. Since for any connected component H of $G - S' - x$ the graph $(G - S')\langle H\rangle$ is a subgraph of $\overline{C} - S'$ or $G[V(C) \cup \{x\}]$ we have that $(G - S')\langle H\rangle$ is outerplanar. Then by Observation 2.2 the graph $G - S'$ is outerplanar.
3. If $|S \cap V(P)| = 1$ then let $u \in S \cap V(P)$ and assume without loss of generality that u lies on the subpath of P from v_3 to v_5, so the subpath of P from v_1 to v_3 does not contain

vertices of S (recall that $v_3 \notin S$). Let $S' := \{v_5\} \cup (S \setminus V(C))$ and note that $|S'| \leq |S|$. We shall show that $G - S'$ is outerplanar. Since $x, v_1 \notin S$, we have that also $x, v_1 \notin S'$, so $xv_1 \in E(G - S')$. In order to apply Lemma 2.3 to $G - S'$ and xv_1 we have to show that

- for each connected component C' of $G - S' - \{v_1, x\}$ the graph $(G - S')\langle C' \rangle$ is outerplanar, and
- there are at most two induced internally vertex-disjoint (v_1, x)-paths in $(G - S') \setminus v_1x$.

Because $v_5 \in S'$ we have $G - S' - \{v_1, x\} = G - \{v_1, v_5, x\} - S'$ and since C is a connected component of $G - \{v_1, v_5, x\}$ we have that all connected components of $G - S' - \{v_1, x\}$ are either a connected component of $C - S' = C$ or of $G - S' - \{v_1, x\} - V(C)$. It is given that C is connected and $G[V(C) \cup \{v_1, v_5, x\}]$ is outerplanar so then $G[V(C) \cup \{v_1, x\}] = (G - S')\langle C \rangle$ is also outerplanar. Any other connected component C' is a connected component of $G - S' - \{v_1, x\} - V(C)$, so we have that $(G - S')\langle C' \rangle$ is a subgraph of $G - S' - V(C)$. This is in turn, a subgraph of $G \setminus xv_3 - S$ which is outerplanar. Hence $(G - S')\langle C' \rangle$ is outerplanar.

It remains to show that there are at most two induced internally vertex-disjoint (v_1, x)-paths in $(G - S') \setminus v_1x$. Suppose for contradiction that $(G - S') \setminus v_1x$ contains three induced vertex-disjoint (v_1, x)-paths. As shown before, C is a connected component of $G - S' - \{v_1, x\}$ adjacent to v_1 and x, so there exists an induced (v_1, x)-path P_1 in $G - S' \setminus v_1x$ whose internal vertices all lie in C. Since $G\langle C \rangle$ is outerplanar and C is connected, by Lemma 2.4 the graph $G\langle C \rangle$ does not contain two vertex-disjoint (v_1, x)-paths with nonempty interiors. Hence there are two induced internally vertex-disjoint (v_1, x)-paths P_2, P_3 in $(G - S' \setminus v_1x) - V(C)$. Observe that P_2 and P_3 are then disjoint from $S \setminus S' \subseteq V(C)$ and do not contain xv_3. It follows that P_1, P_2 and P_3 are three induced internally vertex-disjoint (v_1, x)-paths in $G \setminus xv_3 - S$, contradicting its outerplanarity by Lemma 2.3. We conclude also the second condition of Lemma 2.3 holds for $G - S'$ and the edge v_1x, hence $G - S'$ is outerplanar. ◀

Next, we use the properties of the (k, c, d)-outerplanar decomposition to show that any solution of size at most k contains all but possibly one vertex from $(X_0 \cup X_1) \cap N_G(C)$, where C is a connected component from $G - (X_0 \cup X_1 \cup Z)$. Adding this vertex to C preserves its outerplanarity because (X_0, X_1) is a (k, c)-augmented modulator. We use this fact together with the result from Lemma 3.17 to identify an irrelevant edge, which leads to the following reduction rule:

▶ **Reduction Rule 3.** *Given a (k, c, d)-outerplanar decomposition (X_0, X_1, Z) of a graph G, a vertex $x \in X_0 \cup X_1$, and five vertices $v_1, \ldots, v_5 \in N_G(x) \setminus (X_0 \cup X_1)$ that lie, in order of increasing index, on an induced path P in $G - (X_0 \cup X_1)$ from v_1 to v_5, such that $N_G(x) \cap V(P) = \{v_1, \ldots, v_5\}$. Let C be the component of $G - (X_0 \cup X_1) - \{v_1, v_5\}$ containing $P - \{v_1, v_5\}$. If $V(C) \cap Z = \emptyset$ remove the edge xv_3.*

▶ **Lemma 3.18** (Safeness)**.** *Suppose that Reduction Rule 3 removes the edge $e = xv_3$ from a graph G. If $\mathsf{opd}(G) > k$ then $\mathsf{opd}(G \setminus e) > k$ and if $\mathsf{opd}(G) \leq k$ then $\mathsf{opd}(G \setminus e) = \mathsf{opd}(G)$.*

We show how this reduction rule can be applied to reduce the number of edges between a vertex $x \in X_0 \cup X_1$ and a connected component in $G - (X_0 \cup X_1 \cup Z)$ to a constant, as depicted on Figure 1. This leads to an $\mathcal{O}(k^4)$ bound on $N_G(X_0 \cup X_1)$.

▶ **Lemma 3.19.** *There is a polynomial-time algorithm that, given a (k, c, d)-outerplanar decomposition (X_0, X_1, Z) of a graph G, a vertex $x \in X_0 \cup X_1$ and a component C of $G - (X_0 \cup X_1 \cup Z)$, either applies Reduction Rule 3 or concludes that $|N_G(x) \cap V(C)| \leq 20$.*

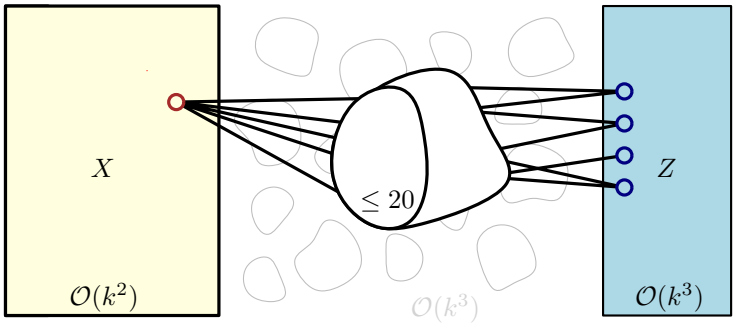


Figure 1 An illustration of Lemma 3.19. Given a (k, c, d)-outerplanar decomposition (X_0, X_1, Z) of a graph G, a vertex $x \in X = X_0 \cup X_1$ and a component C of $G - (X_0 \cup X_1 \cup Z)$, we are guaranteed that $|N(C) \cap Z| \leq 4$ and we can apply Reduction Rule 3 until $|N(x) \cap V(C)| \leq 20$. The expressions at the bottom bound the size of X, the number of components of $G - (X \cup Z)$, and size of Z.

We are going to apply Lemma 3.19 to a computed outerplanar decomposition in order to reduce the total neighborhood size of $X_0 \cup X_1$. This allows us to construct a final modulator L of size $\mathcal{O}(k^4)$ with a structure referred to in previous works as a protrusion decomposition. We can now proceed to stating a lemma that encapsulates application of Reduction Rule 3.

▶ **Lemma 3.20.** *There exists a function $f_5 \colon \mathbb{N}^2 \to \mathbb{N}$ and a polynomial-time algorithm that, given a (k, c, d)-outerplanar decomposition (X_0, X_1, Z) of a graph G, either applies Reduction Rule 2 or Reduction Rule 3, or outputs a set $L \subseteq V(G)$ such that*

1. $|L| \leq f_5(c, d) \cdot (k + 3)^4$,
2. $|E_G(L, L)| \leq f_5(c, d) \cdot (k + 3)^4$,
3. *there are at most $f_5(c, d) \cdot (k + 3)^4$ connected components in $G - L$, and*
4. *for each connected component C of $G - L$ the graph $G\langle C \rangle$ is outerplanar and $|N_G(C)| \leq 4$.*

Furthermore, $f_5(c, d) = 24 \cdot (20 \cdot f_4(c, d) + d + c + c^2)$ (see Lemma 3.15).

4 Compressing the outerplanar subgraphs

After the decomposition of Lemma 3.20, it suffices to apply four reduction rules which shrink outerplanar graphs which connect to the rest of the graph through at most four vertices. We present these rules below. Their correctness proofs are deferred to the full version.

The two rules below shrink outerplanar subgraphs which connect to the rest of the graph via at most two vertices. Both rules yield a minor of the original graph; see Figure 2.

▶ **Reduction Rule 4.** *Consider a graph G and vertex set $C \subseteq V(G)$ such that $N_G(C) = \{x, y\}$, $xy \notin E(G)$, $G[C]$ is connected, and $G\langle C \rangle$ is outerplanar. Let $P = (u_1, u_2, \ldots, u_m)$, $u_1 = x$, $u_m = y$ be any shortest path connecting x and y in $G\langle C \rangle$ and $D_1, D_2, \ldots, D_\ell$ be the connected components of $G\langle C \rangle - V(P)$. We consider 3 cases:*

1. *if there is a component D_i, for which $N_G(D_i)$ includes two non-consecutive elements of P, replace C with two vertices c_1, c_2, each adjacent to both x and y,*
2. *if there are two distinct components D_i, D_j, for which $|N_G(D_i) \cap N_G(D_j)| \geq 2$, replace C with two vertices c_1, c_2, each adjacent to both x and y,*
3. *otherwise replace C with one vertex c_1 adjacent to both x and y.*

▶ **Reduction Rule 5.** *Suppose that there is an edge $e = uv$ in a graph G such that $G - V(e)$ has a connected component C such that $G\langle C \rangle$ is outerplanar. Then contract C into a single vertex.*

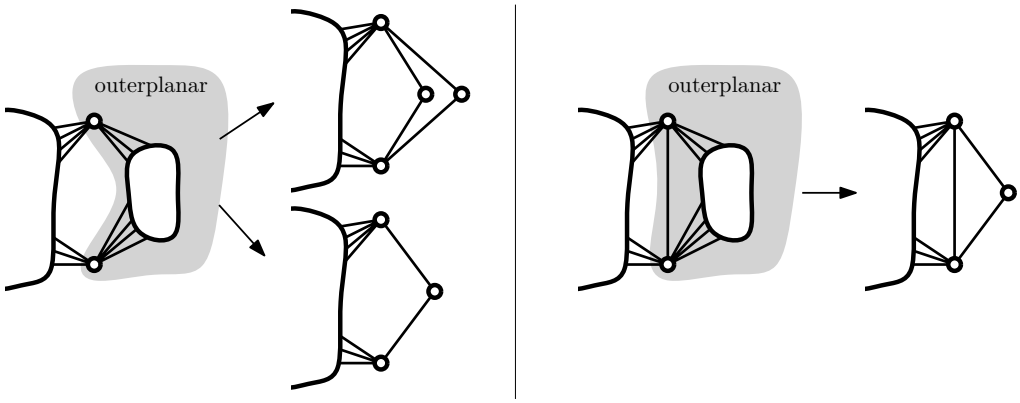


Figure 2 On the left a depiction of Reduction Rule 4, which reduces a connected subgraph to one or two vertices depending on its internal structure. On the right a depiction of Reduction Rule 5 which contracts a connected subgraph to a single vertex if it is outerplanar together with the two adjacent vertices that form its neighborhood.

The following reduction rule targets fan structures in outerplanar subgraphs. Its safeness follows directly from Lemma 3.17.

▶ **Reduction Rule 6.** *Suppose we are given a graph G, a vertex $x \in V(G)$, and five vertices $v_1, \ldots, v_5 \in N_G(x)$ that lie, in order of increasing index, on an induced path P in $G - x$ from v_1 to v_5, such that $N_G(x) \cap V(P) = \{v_1, \ldots, v_5\}$. Let C be the component of $G - \{v_1, v_5, x\}$ containing $P - \{v_1, v_5\}$. If $G\langle C\rangle$ is outerplanar, then remove the edge xv_3.*

The final reduction rule reduces ladder structures in biconnected outerplanar graphs. For its statement, we need the following terminology.

▶ **Definition 4.1.** *For a graph G, a sequence of edges $e_1, \ldots, e_\ell \in E(G)$ is an* order-respecting matching *if the set of edges is a matching and if for all $1 \leq i < j < k \leq \ell$ we have that e_i and e_k are in different connected components of $G - V(e_j)$.*

▶ **Reduction Rule 7.** *Let G be a graph, $e_1, \ldots, e_7$ be a matching in G, and let C be a connected component of $G - (V(e_1) \cup V(e_7))$. If $\{e_2, \ldots, e_6\} \subseteq E(C, C)$, $N_G(C) = V(e_1) \cup V(e_7)$, $G\langle C\rangle$ is biconnected and outerplanar, and $e_1, \ldots, e_7$ is an order-respecting matching in $G\langle C\rangle$, then remove e_4.*

The last two reduction rules are depicted on Figure 3. Intuitively, Reduction Rule 4 and Reduction Rule 5 together with the earlier stated Reduction Rule 2 reduce the number of biconnected components in an outerplanar graph $G\langle C\rangle$ with $|N_G(C)| \leq 4$ to a constant number (26). We show that if any biconnected component is large, then either it contains a large face so that Reduction Rule 4 can be applied to two vertices along the face that cut off a large outerplanar subgraph, it contains a large outerplanar subgraph attached onto an edge so that Reduction Rule 5 can be applied, or it contains a large fan (Reduction Rule 6) or ladder (Reduction Rule 7) structure that contains an irrelevant edge.

5 Wrapping up

The following lemma summarizes the effect of the four reduction rules described in Section 4.

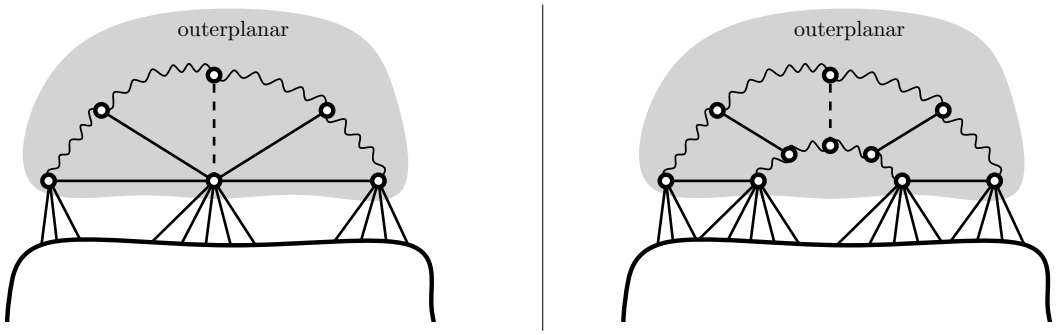


Figure 3 On the left a depiction of Reduction Rule 6 which is able to remove the middle edge of a fan structure in an outerplanar subgraph that is sufficiently isolated from the rest of the graph. On the right a depiction of Reduction Rule 7, which removes the middle edge of an order-respecting matching in an outerplanar subgraph that is sufficiently isolated from the rest of the graph.

▶ **Lemma 5.1.** *Consider a graph G and a vertex set $A \subseteq V(G)$, such that $|A| > 25 \cdot 6288$, $|N_G(A)| \leq 4$, $G[A]$ is connected, and $G\langle A\rangle$ is outerplanar. There is a polynomial-time algorithm that, given G and A satisfying the conditions above, outputs a proper minor G' of G, so that $\mathsf{opd}(G') = \mathsf{opd}(G)$.*

We repeatedly apply this reduction using the decomposition given by the set $L \subseteq V(G)$ of Lemma 3.20. It is important that the graph is guaranteed to shrink at each step, so after polynomially many invocations of Lemma 5.1 we must arrive at an irreducible instance. We now state the main theorem with the final bound on the size of compressed graph $2 \cdot (25 \cdot 6288 + 5) \cdot f_5(c, f_3(c)) \cdot (k_2 + 3)^4$ (see Lemmas 3.14 and 3.20), where $c = 40$. Recall that instances (G, k) and (G', k') are equivalent if $\mathsf{opd}(G) \leq k \Leftrightarrow \mathsf{opd}(G') \leq k'$.

▶ **Theorem** (1.1, restated)**.** *The Outerplanar Deletion problem admits a polynomial-time kernelization algorithm that, given an instance (G, k), outputs an equivalent instance (G', k'), such that $k' \leq k$, graph G' is a minor of G, and G' has $\mathcal{O}(k^4)$ vertices and edges. Furthermore, if $\mathsf{opd}(G) \leq k$, then $\mathsf{opd}(G') = \mathsf{opd}(G) - (k - k')$.*

As a consequence of the theorem above, we obtain the first concrete bounds on the sizes of minor-minimal obstructions to having an outerplanar vertex deletion set of size k.

▶ **Corollary** (1.2, restated)**.** *If G is a graph such that $\mathsf{opd}(G) > k$ but each proper minor G' of G satisfies $\mathsf{opd}(G') \leq k$, then G has $\mathcal{O}(k^4)$ vertices and edges.*

6 Conclusion

We presented a number of elementary reduction rules for Outerplanar Deletion that can be applied in polynomial time to obtain a kernel of $\mathcal{O}(k^4)$ vertices and edges. This kernel does not use protrusion replacement and the constants hidden by the $\mathcal{O}$-notation can be derived easily. This is the first concrete kernel for Outerplanar Deletion, and a step towards more concrete kernelization bounds for Planar-$\mathcal{F}$ Deletion. We hope it inspires new kernelization bounds for Planar Deletion.

In earlier work Dell and Van Melkebeek [9, Theorem 3] have shown that there is no kernel for Outerplanar Deletion of bitsize $\mathcal{O}(k^{2-\varepsilon})$ unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$. This naturally leads to the question, can these two bounds be brought closer together?

Another interesting direction for further research is to obtain concrete kernelization bounds for other Planar-$\mathcal{F}$ Deletion problems. Our work exploits the fact that $K_{2,3}$-minor-free graphs cannot have many disjoint paths between two vertices. Previous work [17] used a similar observation to derive a kernel for θ_c-Minor-Free Deletion. An interesting next case would be a Planar-$\mathcal{F}$ Deletion problem where $\mathcal{F}$ does not contain $K_{2,c}$ or θ_c for some c, for example 2-Transversal which asks whether a graph of treewidth at most 2 can be obtained by deleting k vertices.

References

1 Therese C. Biedl. Small drawings of outerplanar graphs, series-parallel graphs, and other planar graphs. *Discret. Comput. Geom.*, 45(1):141–160, 2011. doi:10.1007/s00454-010-9310-z.

2 Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (Meta) Kernelization. *J. ACM*, 63(5):44:1–44:69, 2016. doi:10.1145/2973749.

3 Kevin Cattell, Michael J. Dinneen, Rodney G. Downey, Michael R. Fellows, and Michael A. Langston. On computing graph minor obstruction sets. *Theor. Comput. Sci.*, 233(1-2):107–127, 2000. doi:10.1016/S0304-3975(97)00300-9.

4 Gary Chartrand and Frank Harary. Planar permutation graphs. *Annales de l'I.H.P. Probabilités et statistiques*, 3(4):433–438, 1967.

5 Jianer Chen, Iyad A. Kanj, and Weijia Jia. Vertex cover: Further observations and further improvements. *Journal of Algorithms*, 41(2):280–301, 2001. doi:10.1006/jagm.2001.1186.

6 David Coudert, Florian Huc, and Jean-Sébastien Sereni. Pathwidth of outerplanar graphs. *J. Graph Theory*, 55(1):27–41, 2007. doi:10.1002/jgt.20218.

7 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*. Springer, 2015.

8 Marek Cygan, Marcin Pilipczuk, Michał Pilipczuk, and Jakub Onufry Wojtaszczyk. An improved FPT algorithm and a quadratic kernel for pathwidth one vertex deletion. *Algorithmica*, 64(1):170–188, September 2012.

9 Holger Dell and Dieter Van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. *J. ACM*, 61(4), July 2014. doi:10.1145/2629620.

10 Guoli Ding and Stan Dziobiak. Excluded-minor characterization of apex-outerplanar graphs. *Graphs Comb.*, 32(2):583–627, 2016. doi:10.1007/s00373-015-1611-9.

11 Michael J. Dinneen. Too many minor order obstructions. *J. Univers. Comput. Sci.*, 3(11):1199–1206, 1997. doi:10.3217/jucs-003-11-1199.

12 Michael J. Dinneen, Kevin Cattell, and Michael R. Fellows. Forbidden minors to graphs with small feedback sets. *Discret. Math.*, 230(1-3):215–252, 2001. doi:10.1016/S0012-365X(00)00083-2.

13 Michael J. Dinneen and Liu Xiong. Minor-order obstructions for the graphs of vertex cover 6. *J. Graph Theory*, 41(3):163–178, 2002. doi:10.1002/jgt.10059.

14 Huib Donkers, Bart M. P. Jansen, and Michał Włodarczyk. Preprocessing for outerplanar vertex deletion: An elementary kernel of quartic size, 2021. arXiv:2110.01868.

15 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.

16 Herbert J. Fleischner, Dennis P. Geller, and Frank Harary. Outerplanar graphs and weak duals. *Journal of the Indian Mathematical Society*, 38, 1974. URL: http://www.informaticsjournals.com/index.php/jims/article/view/16694.

17 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, Geevarghese Philip, and Saket Saurabh. Hitting forbidden minors: Approximation and kernelization. *SIAM J. Discret. Math.*, 30(1):383–410, 2016. doi:10.1137/140997889.

18 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar $\mathcal{F}$-deletion: Approximation, kernelization and optimal FPT algorithms. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 470–479. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.62.

19 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019. doi:10.1017/9781107415157.

20 Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: theory of parameterized preprocessing*. Cambridge University Press, 2019.

21 Fabrizio Frati. Straight-line drawings of outerplanar graphs in $O(dn \log n)$ area. *Computational Geometry*, 45(9):524–533, 2012. doi:10.1016/j.comgeo.2010.03.007.

22 Emilio Di Giacomo, Giuseppe Liotta, and Tamara Mchedlidze. Lower and upper bounds for long induced paths in 3-connected planar graphs. *Theor. Comput. Sci.*, 636:47–55, 2016. doi:10.1016/j.tcs.2016.04.034.

23 Archontia C. Giannopoulou, Bart M. P. Jansen, Daniel Lokshtanov, and Saket Saurabh. Uniform kernelization complexity of hitting forbidden minors. *ACM Trans. Algorithms*, 13(3), March 2017. doi:10.1145/3029051.

24 Anupam Gupta, Euiwoong Lee, Jason Li, Pasin Manurangsi, and Michał Włodarczyk. Losing treewidth by separating subsets. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1731–1749. SIAM, 2019. doi:10.1137/1.9781611975482.104.

25 Yoichi Iwata. Linear-Time Kernelization for Feedback Vertex Set. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 68:1–68:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2017.68.

26 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.

27 Bart M. P. Jansen and Marcin Pilipczuk. Approximation and kernelization for chordal vertex deletion. *SIAM J. Discret. Math.*, 32(3):2258–2301, 2018. doi:10.1137/17M112035X.

28 Gwenaël Joret, Christophe Paul, Ignasi Sau, Saket Saurabh, and Stéphan Thomassé. Hitting and harvesting pumpkins. *SIAM J. Discret. Math.*, 28(3):1363–1390, 2014. doi:10.1137/120883736.

29 Jens Lagergren. Upper bounds on the size of obstructions and intertwines. *J. Comb. Theory, Ser. B*, 73(1):7–40, 1998. doi:10.1006/jctb.1997.1788.

30 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.

31 J. Leydold and P. Stadler. Minimal cycle bases of outerplanar graphs. *Electron. J. Comb.*, 5, 1998.

32 Tamara Mchedlidze and Antonios Symvonis. Crossing-optimal acyclic hp-completion for outerplanar st-digraphs. *J. Graph Algorithms Appl.*, 15(3):373–415, 2011. doi:10.7155/jgaa.00231.

33 Kerri Morgan and Graham Farr. Approximation algorithms for the maximum induced planar and outerplanar subgraph problems. *J. Graph Algorithms Appl.*, 11(1):165–193, 2007. doi:10.7155/jgaa.00141.

34 Geevarghese Philip, Venkatesh Raman, and Yngve Villanger. A quartic kernel for pathwidth-one vertex deletion. In Dimitrios M. Thilikos, editor, *Graph Theoretic Concepts in Computer Science - 36th International Workshop, WG 2010, Zarós, Crete, Greece, June 28-30, 2010 Revised Papers*, volume 6410 of *Lecture Notes in Computer Science*, pages 196–207, 2010. doi:10.1007/978-3-642-16926-7_19.

35 Timo Poranen. Heuristics for the maximum outerplanar subgraph problem. *J. Heuristics*, 11(1):59–88, 2005. doi:10.1007/s10732-005-6999-6.

36 Neil Robertson and Paul D. Seymour. Graph minors. V. Excluding a planar graph. *J. Comb. Theory, Ser. B*, 41(1):92–114, 1986. doi:10.1016/0095-8956(86)90030-4.

37 Juanjo Rué, Konstantinos S. Stavropoulos, and Dimitrios M. Thilikos. Outerplanar obstructions for a feedback vertex set. *Eur. J. Comb.*, 33(5):948–968, 2012. doi:10.1016/j.ejc.2011.09.018.

38 Ignasi Sau, Giannos Stamoulis, and Dimitrios M. Thilikos. An FPT-algorithm for recognizing k-apices of minor-closed graph classes. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 95:1–95:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.95.

39 Ignasi Sau, Giannos Stamoulis, and Dimitrios M. Thilikos. k-apices of minor-closed graph classes. I. Bounding the obstructions. *CoRR*, abs/2103.00882, 2021. arXiv:2103.00882.

40 Saket Saurabh. Open problems from the workshop on kernelization (WorKer 2019), 2019. URL: https://www.youtube.com/watch?v=vCjG5zGjQr4.

41 Maciej M. Syslo. Characterizations of outerplanar graphs. *Discret. Math.*, 26(1):47–53, 1979. doi:10.1016/0012-365X(79)90060-8.

42 Stéphan Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM Trans. Algorithms*, 6(2):32:1–32:8, 2010. doi:10.1145/1721837.1721848.

43 René Van Bevern, Hannes Moser, and Rolf Niedermeier. Approximation and tidying—a problem kernel for s-plex cluster vertex deletion. *Algorithmica*, 62(3):930–950, 2012.

Maximum Matching in Almost Linear Time on Graphs of Bounded Clique-Width

Guillaume Ducoffe
National Institute for Research and Development in Informatics, Bucharest, Romania
University of Bucharest, Romania

Abstract

Recently, independent groups of researchers have presented algorithms to compute a maximum matching in $\tilde{\mathcal{O}}(f(k) \cdot (n+m))$ time, for some computable function f, within the graphs where some clique-width upper bound is at most k (*e.g.*, tree-width, modular-width and P_4-sparseness). However, to the best of our knowledge, the existence of such algorithm within the graphs of bounded clique-width has remained open until this paper. Indeed, we cannot even apply Courcelle's theorem to this problem directly, because a matching cannot be expressed in MSO_1 logic.

Our first contribution is an almost linear-time algorithm to compute a maximum matching in any bounded clique-width graph, being given a corresponding clique-width expression. It can be used to also compute the Edmonds-Gallai decomposition. For that, we do apply Courcelle's theorem, but in order to compute the cardinality of a maximum matching rather than the matching itself, via the classic Tutte-Berge formula. To obtain with this approach a maximum matching, we need to combine it with a recursive dissection scheme for bounded clique-width graphs based on the existence of balanced edge-cuts with bounded neighbourhood diversity, and with a distributed version of Courcelle's theorem (Courcelle and Vanicat, *DAM 2016*) – of which we present here a slightly stronger version than the standard one in the literature – in order to evaluate the Tutte-Berge formula on various subgraphs of the input.

Finally, for the bipartite graphs of clique-width at most k, we present an alternative $\tilde{\mathcal{O}}(k^2 \cdot (n+m))$-time algorithm for the problem. The algorithm is randomized and it is based on a completely different approach than above: combining various reductions to matching and flow problems on bounded tree-width graphs with a very recent result on the parameterized complexity of linear programming (Dong et. al., *STOC*'21). Our results for bounded clique-width graphs extend many prior works on the complexity of Maximum Matching within cographs, distance-hereditary graphs, series-parallel graphs and other subclasses.

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

Keywords and phrases Maximum Matching, Maximum b-matching, Clique-width, Tree-width, Courcelle's theorem, FPT in P

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.15

Funding This work was supported by Grant TC ICUB-SSE 15109-26.07.2021, "The complexity landscape of Maximum Matching". It was also supported by project PN-19-37-04-01 "New solutions for complex problems in current ICT research fields based on modelling and optimization", funded by the Romanian Core Program of the Ministry of Research and Innovation (MCI) 2019-2022.

1 Introduction

For any undefined graph terminology, see [7, 23]. Throughout the paper, for any graph $G = (V, E)$, let $n := |V|$ be its *order* (number of vertices) and $m := |E|$ be its *size* (number of edges). Recall that a *matching* in a graph is a set of pairwise end-disjoint edges. A *maximum matching* is one of maximum cardinality. The *matching number* of G, denoted by $\nu(G)$, is the cardinality of a maximum matching of G. Matchings (possibly, with additional constraints) are ubiquitous in scheduling, markets, resource allocation schemes and even chemistry [68]. We refer to [44, 55] for a compendium of matching problems and their applications.

This paper is about the (parameterized) complexity of Maximum Matching in graphs. Unsurprisingly, a lot of research in Computer Science has been devoted to this question. The first polynomial-time algorithm for Maximum Matching was proposed by Edmonds [30]. Later, Micali and Vazirani presented the state-of-the-art $\mathcal{O}(m\sqrt{n})$-time algorithm for this problem [60], that has remained unchallenged since forty years. We here study Maximum Matching in the context of "Fine-Grained Complexity in P" (see [72] for a survey of this blossoming field). Specifically, *can a maximum matching be computed in (almost) linear time?* There are a few reasons to believe that it is indeed the case. For instance, unlike for the diameter problem and other fundamental graph problems, for which over the last decades, conditional superlinear lower bounds were obtained, it is known [8] that proving such lower bound for Maximum Matching would falsify the so-called Nondeterministic Strong Exponential Time Hypothesis (NSETH). Furthermore, computing a maximum matching is related to Maximum Flow [71], that is sometimes conjectured to be solvable in linear time.

The idea of using tools and concepts from parameterized complexity in the context of polynomial-time solvable problems has been scarce [48]. In part motivated by the recent "SETH-hardness" results, and other conditional lower bounds for such problems [73], a richer theory of "FPT in P" has started to emerge recently [1, 5, 32, 36, 45]. In its simplest form, the former is about the existence of $\mathcal{O}(f(k)\cdot(n+m)^{1+o(1)})$-time algorithms for various graph problems when some fixed parameter is at most k [1]. As far as we are concerned here, such running times were obtained in [13, 29, 28, 36, 45, 49, 50, 51, 59] for Maximum Matching, for different parameterizations. For instance, for the graphs of *tree-width* at most k, Fomin et. al [36] presented a randomized $\tilde{\mathcal{O}}(k^4\cdot n)$-time algorithm for computing a maximum matching [2]. This was later improved by Iwata et. al. [50], who designed a deterministic $\tilde{\mathcal{O}}(k^2\cdot n)$-time algorithm for that problem. – We recall the definition of tree-width in Sec. 2.2. – Remarkably, the parameterized study of Maximum Matching has led to the development of many nice techniques in this area, which brought Mertzios et. al. [59] to nickname the problem the "drosophilia" of the study of the FPT algorithms in P.

Clique-width is one of the most studied graph parameters. It is a rough estimate of the closeness of a graph to a cograph (*a.k.a.*, P_4-free graph). We refer to Sec. 2.1 for a formal definition. Note that unlike for the tree-width, there exist dense graphs of bounded clique-width (*e.g.*, the complete graphs and the complete bipartite graphs). The applications of clique-width to NP-hard problems, including Courcelle's theorem [15] and some general algorithmic frameworks [31], are now rather well understood [33, 34, 35]. However, the study of its applications to polynomial-time solvable problems is comparatively much more recent and, so far, limited to cycle problems [5, 13] and distance problems [13, 16, 27, 52]. Parameterized almost linear-time algorithms for Maximum Matching are known for the important subclasses of bounded tree-width graphs [36, 50], graphs of bounded modular-width [13, 51], and some others [13, 28, 29]. However, as far as we know, the complexity of this problem on bounded clique-width graphs has been open until this article. Indeed we stress that, even allowing a super-polynomial dependency on the clique-width in the running time, the existence of an almost linear-time (parameterized) algorithm for Maximum Matching does not follow from Courcelle's theorem, because a matching cannot be expressed in MSO_1 logic. This is in sharp contrast with bounded tree-width graphs, for which we

[1] More generally, the goal is, for some problem solvable in $\mathcal{O}(m^{q+o(1)})$ time on arbitrary m-edge graphs, to design an $\mathcal{O}(f(k)m^{p+o(1)})$-time algorithm, for some $p<q$, within the class of graphs where some fixed parameter is at most k.

[2] The $\tilde{\mathcal{O}}()$ notation suppresses poly-logarithmic factors.

can apply Courcelle's theorem for the stronger MSO_2 logic (allowing quantification over subsets of edges), and so, in particular in order to express a matching [18]. – We refer to Sec. 3 for a reminder about MSO logic. – Furthermore if we consider the related problem Maximum-Weight Matching, then it has been observed [51] that it is as hard on bounded clique-width weighted graphs as on general weighted graphs under $\mathcal{O}(n^2)$-time reductions. Again, this differs from the case of bounded tree-width graphs, for which an $\tilde{\mathcal{O}}(k^2n)$-time algorithm also exists for this problem [50].

Beyond the study of the FPT algorithms in P, it also makes sense to study Maximum Matching on restricted graph classes, both as a way to better understand the hard instances for this problem, and to better model some of its real-life applications (see [46] for an example of the latter). In this respect, a considerable amount of positive results have been proved [9, 22, 26, 38, 37, 46, 54, 58, 61, 74, 76, 75]. Many such classes, starting from the cographs [20], are known to have bounded clique-width. Therefore, having at hands an almost linear-time algorithm for Maximum Matching on bounded clique-width graphs, one can unify and generalize many prior works in this area.

1.1 Our results

Recall that a graph has clique-width at most k if and only if it admits a *k-expression* [20]. Such a k-expression can be computed in linear time on many interesting subclasses of bounded clique-width graphs [3]: ranging from cographs [20], switched cographs [11], distance-hereditary graphs [47], $(q, q-3)$-graphs [57], and graphs of either bounded tree-width [12], modular-width [20] or split-width [65].

Hereafter, we use the notation $\tilde{\mathcal{O}}_{x_1,x_2,\ldots,x_t}(n+m)$ for a running time in $\tilde{\mathcal{O}}(f(x_1, x_2, \ldots, x_t)\cdot(n+m))$, for some computable function f. The following is our first main result in the paper:

▶ **Theorem 1.** *Given a graph G and a corresponding k-expression, one can compute a maximum matching for G in deterministic $\tilde{\mathcal{O}}_k(n+m)$ time.*

To the best of our knowledge, this is the first almost linear-time algorithm for Maximum Matching on bounded clique-width graphs. The $\tilde{\mathcal{O}}_k()$ notation hides huge constants in k due to our use of Courcelle's theorem. Indeed, while we cannot express a matching in MSO_1 logic, we can write a *Counting* MSO_1 formula in order to evaluate the matching number (Theorem 6). For that, we use the well-known *Tutte-Berge* formula [6]. This alone does not lead to an efficient computation of a maximum matching, but only of its size. However, by carefully evaluating our formula for the matching number on various subgraphs, obtained by removing specific vertex- and edge-subsets, one can compute a maximum matching incrementally. A similar approach also works for computing the *Edmonds-Gallai decomposition* [30, 42, 43], which somehow encodes the structure of all the maximum matchings in a graph (Theorem 8). The main difficulty here is that the number of subgraphs on which we need to evaluate our formula is not constant. Thus, applying Courcelle's theorem to each subgraph separately would result in a super-linear running time. We overcome this issue by using a distributed version of this theorem [17]. In doing so, after we computed the matching number of a bounded clique-width graph G in almost linear time, it becomes possible to evaluate our formula on any subgraph H in time roughly proportional to the number of basic operations needed to obtain H from G.

[3] So far, the best-known approximation algorithms for clique-width run in $\mathcal{O}(n^3)$-time, that is slower than the state-of-the-art algorithm for Maximum Matching [62].

It seems that improving the dependency on k in the running time will require new techniques. Our second main result is that it can be done for *bipartite* graphs of bounded clique-width:

▶ **Theorem 2.** *Given a bipartite graph G and a corresponding k-expression, one can compute a maximum matching for G in randomized $\tilde{\mathcal{O}}(k^2 \cdot (n+m))$ time.*

Let us sketch below the main lines of our approach toward proving Theorem 2. We first reduce Maximum Matching on bounded clique-width graphs to a related problem on the graphs of bounded *tree-width*. The reduction preserves the property for a graph to be bipartite. Its intuition goes as follows. Roughly, graphs of bounded tree-width can be recursively disconnected by some small balanced vertex-separators. By comparison, graphs of bounded clique-width can be recursively disconnected by some balanced edge-cuts of small "neighbourhood diversity" (partitionable in a small number of complete joins) [19]. To reduce to the bounded tree-width case, we propose a transformation of edge-cuts of small neighbourhood diversity into small vertex-separators (Sec. 5.1). The transformation forces us to deal with a more general problem than Maximum Matching, sometimes called Maximum b-Matching and well-studied on its own [4, 40, 41, 56, 63, 64]. We thus exchange Maximum Matching for a more complex problem, but on a structurally simpler graph class. Furthermore, because we restrict ourselves to bipartite graphs, we can solve Maximum b-Matching as a linear program. To the matrix representation of any such linear program, one can associate various graphs. Then, it becomes possible to define the tree-width of a linear program. In [36], Fomin et. al. asked whether all linear programs of bounded tree-width could be solved in almost linear time. Very recently, Dong et. al. gave a positive answer [25]. – This is where we need randomization. – We apply this nice result to the problem Maximum b-Matching within bipartite graphs. Here, some final technicalities arise due to the algorithm of Dong et. al. only outputting an approximate fractional b-matching whereas we aim at computing an exact integral solution. This can be overcome by using the close connection between Maximum Flow and Maximum b-Matching on bipartite graphs, along with a nice result from Madry to apply rounding to a fractional flow [56].

1.2 Related work

The are several meta-theorems deduced from Courcelle's theorem in the literature. Indeed, Courcelle's approach not only applies to decision problems, but also to counting [14] and optimization problems [15]. We actually use in our proof the optimization version of his theorem. Applications to the design of distance-labelling schemes were proposed in [17], and later refined in [16, 27] using alternative techniques. However, insofar most applications of Courcelle's theorem are about NP-hard problems. Indeed, Abboud et. al. [1] observed that its use leads to huge dependencies on the parameter involved, that can often be sharpened by preferring other techniques (their observation, on the other hand, also remains valid for NP-hard problems). What we find intriguing in our case is, first, the nontrivial use we need to make of Courcelle's theorem for a polynomial-time solvable problem, and second, that we currently do not see any other way to obtain a quasi linear-time algorithm for Maximum Matching on the bounded clique-width graphs. This is evidence, we believe, that Courcelle's theorem could help in expanding the nascent field of "FPT in P".

The proof of our Theorem 1 also has several aspects that, we think, are equally intriguing. For one, we avoid computing augmenting paths, and we do not need the Tutte matrix [70] either. Both concepts are the cornerstone of almost all maximum matching algorithms in the literature. To the best of our knowledge, our result is one of the very first algorithmic applications of the Tutte-Berge formula. The latter also got used in [8], but the algorithm

in this related work was non-deterministic. Our repeated use of this formula in order to compute a maximum matching is not unlike the celebrated result of Anari and Vazirani that reduces the efficient parallel computation of such matching to the design of an oracle for a decision version of the problem [3]. Nevertheless, both results have fairly different proofs.

About Theorem 2, we note that different reductions from Maximum Matching to Maximum b-Matching have already been considered for graphs of bounded modular-width [51] or bounded split-width [28], that are subclasses of bounded clique-width graphs. However, from the algorithmic point of view, the instances of Maximum b-Matching outputted by these former reductions are of bounded size, a much more restricted case than bounded tree-width. To our best knowledge, the Maximum b-Matching problem has only been solved in almost linear time on subclasses of graphs of tree-width at most two [29]. We left open the parameterized complexity of Maximum b-Matching within the graphs of bounded tree-width. For general graphs, the so-called "Russian method" starts from the linear relaxation of this problem (written as an integer program) and it repeatedly adds "blossom constraints" that are violated by the current solution until it finds an optimal integral outcome [63]. These blossom constraints are deduced from the good characterization of the b-matching polytope by Edmonds and Pulleyblank [64]. It seems, however, that a super-linear (but polynomial) number of linear programs needs to be solved on general graphs. See also Anstee [4] and Gabow [40] for alternative algorithms.

1.3 Organization of the paper

In Sec. 2, we introduce some basic notations and terminology, as well as the two graph parameters considered in this article. Sec. 3 is devoted to Courcelle's theorem for bounded clique-width graphs. We need a distributed version of this theorem, for optimization functions, of which a proof by Courcelle and Vanicat can be found in [17] but, unfortunately, for a more restricted setting than what we need. While it is not excessively difficult to check that Courcelle and Vanicat's proof indeed works in the broader setting that is here needed, the proof is fairly long and it has several intermediate steps, which is why we found it better to write it down almost completely. Then, in Sec. 4, we apply this result in order to compute the matching number, a corresponding maximum matching, and the Edmonds-Gallai decomposition, within bounded clique-width graphs. Sec. 5 is devoted to the proof of Theorem 2. We then conclude in Sec. 6.

2 Preliminaries

First we complete the basic graph terminology given in Sec. 1. By a graph, we mean a finite, simple, unweighted undirected graph. Let $G = (V, E)$ be such a graph. The (open) *neighbourhood* of a vertex v is defined as $N_G(v) := \{u \in V \mid uv \in E\}$. Its *closed neighbourhood* is defined as $N_G[v] := N_G(v) \cup \{v\}$. Let $d_G(v) := |N_G(v)|$ be the *degree* of vertex v. Similarly, for a vertex-subset S, let $N_G(S) := \bigcup_{v \in S} N_G(v) \setminus S$ denote its (open) neighbourhood and let $N_G[S] := S \cup N_G(S)$ denote its closed neighbourhood. We sometimes omit the subscript if the graph G is clear from the context.

2.1 Clique-width

A *k-labeled graph* is a triple $G = (V, E, \ell)$ where $\ell : V \to \{1, 2, \ldots, k\}$ is called a labeling function. A k-expression is an algebraic expression where the four allowed operations are:

- $i(v)$: we add a new isolated vertex with label $\ell(v) = i$;
- $G_1 \oplus G_2$: we make the disjoint union of two k-labeled graphs;

- $\eta(i,j)$: we add a join (complete bipartite subgraph) between all vertices with label i and all vertices with label j;
- $\rho(i,j)$: for all vertices v s.t. $\ell(v) = i$, we set $\ell(v) = j$.

The *generated graph* is the one obtained from the k-expression by deleting all the labels. The clique-width of a graph G, denoted by $cw(G)$, is the least k such that it is the graph generated by some k-expression.

It is useful to see a k-expression as a rooted tree. Namely, the leaves of this tree are labeled by the operations $i(v)$ for vertex-creation. The internal nodes are labeled by the other operations, with the degree of each such node being the arity of the corresponding operation: 1 for the join operations and the relabeling operations, and 2 for the disjoint union operations. See Fig. 1 for an example. We call this tree representation the *parse tree* of the k-expression.

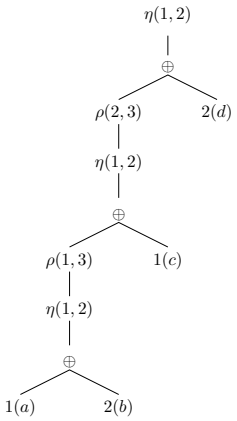


Figure 1 The parse tree of some 3-expression of P_4.

The size of a k-expression is its number of operations (= number of nodes in its parse tree). Throughout the remainder of the paper, we assume each given k-expression for a graph to be of linear size $\mathcal{O}(n+m)$. Note that it is always the case if we restrict ourselves to a subclass where a k-expression can be computed in linear time, that is anyway the relevant case for which our results in this paper could be applied. More generally, any k-expression can be transformed into an equivalent k-expression of size $\mathcal{O}(n+m)$ [39].

2.2 Tree-width

A *tree decomposition* $(T, \mathcal{X})$ of $G = (V, E)$ is a pair consisting of a tree T and of a family $\mathcal{X} = (X_t)_{t \in V(T)}$ of subsets of V indexed by the nodes of T and satisfying:

- $\bigcup_{t \in V(T)} X_t = V$;
- for any edge $e = \{u, v\} \in E$, there exists $t \in V(T)$ such that $u, v \in X_t$;
- for any $v \in V$, the set of nodes $\{t \in V(T) \mid v \in X_t\}$ induces a subtree T_v of T.

The sets X_t are called *the bags* of the decomposition. The *width* of a tree decomposition is the size of a largest bag minus one. Finally, the *tree-width* of a graph G, denoted by $\mathsf{tw}(G)$, is the least possible width over its tree decompositions. We only use tree-width in Sec. 5.

3 Courcelle's theorem

We refer to [14] for a thorough treatment of graph logics and their algorithmic applications. Recall that in *monadic second-order logic* (for short, MSO logic), we are given first-order variables x (written in lower-case), and set variables X (written in upper-case). We allow

atomic formulas of the form $x \in X$, expressing the membership of x to a set X. The *counting MSO* (for short, $CMSO$) further allows atomic formulas of the form $Card_{p,q}(X)$, expressing that $|X| \equiv p \pmod q$ and sometimes called *counting predicates*. – We here assume p and q to be fixed constants, but it should be noted that in the general case, the complexity of $CMSO$ model checking also depends on the values of p and q. – The $CMSO$ logic is stronger than MSO logic: for instance, there is no MSO formula expressing that a set has even cardinality [14]. In Sec. 4, we will need the counting predicates of $CMSO$ logic in order to express the Tutte-Berge formula. Before that, we need to define $CMSO$ *optimization functions*.

Let φ be some $CMSO$ formula with $r + s$ free variables, and let $a_1, a_2, \ldots, a_s$ be fixed integers – possibly, null or negative. Let $Z_1, Z_2, \ldots, Z_r$ be fixed subsets of the domain of the first-order variables. We define $\psi(Z_1, Z_2, \ldots, Z_r)$ as the minimum value $\sum_{i=1}^{s} a_i \cdot |X_i|$ amongst all subsets $X_1, X_2, \ldots, X_s$ such that $\varphi(Z_1, Z_2, \ldots, Z_r, X_1, X_2, \ldots, X_s)$ is true. Then, ψ is a $CMSO$ optimization function of arity r. We define the size of ψ as $|\psi| = r + s$ (*a.k.a.*, as the arity of the underlying $CMSO$ formula φ).

To a c-labelled graph $G = (V, E, \ell)$, we can associate the *relational structure* $\langle V, \{inc, lab_1, lab_2, \ldots, lab_c\}\rangle$ where the vertex-set V is the domain of first-order variables, the binary operator $inc : V \times V \to \{0, 1\}$ asserts whether two vertices are adjacent in G and, for each i, $lab_i : V \to \{0, 1\}$ asserts whether the label of a given vertex equals i. The $(C)MSO_1$ logic on graphs is the above $(C)MSO$ logic restricted to such structures. We define $CMSO_1$ optimization functions in the exact same way. There is a more general $(C)MSO_2$ logic, where we also allow variables to represent edges, but it is not discussed here. Finally, define the underlying graph of G as the graph obtained from G by removing all its labels.

The following result was proved by Courcelle and Vanicat [17], but only for MSO_1 logic and for a more restricted type of optimization problems.

▶ **Theorem 3.** *Let ψ be a $CMSO_1$ optimization function on c-labelled graphs, for some fixed constant c, and of arity r. For every c-labelled graph G of clique-width at most k, if a k-expression is given for the underlying graph of G then, after a pre-processing in $\tilde{\mathcal{O}}_{k,|\psi|}(n + m)$ time, for every vertex-subsets $Z_1, Z_2, \ldots, Z_r$ of G, we can compute the value $\psi(Z_1, Z_2, \ldots, Z_r)$ in $\tilde{\mathcal{O}}_{k,|\psi|}\left(\sum_{j=1}^{r} |Z_j|\right)$ time.*

As a particular case of the above Theorem 3 (for $r = 0$), we retrieve the optimization version of Courcelle's theorem for bounded clique-width graphs (see [15]):

▶ **Theorem 4.** *Let φ be a $CMSO_1$ formula on c-labelled graphs, for some fixed constant c, and with s free variables. Let. also $a_1, a_2, \ldots, a_s$ be fixed integers. For every c-labelled graph G of clique-width at most k, if a k-expression is given for the underlying graph of G then, in $\tilde{\mathcal{O}}_{k,|\varphi|}(n + m)$ time, one can compute the minimum value $\sum_{i=1}^{s} a_i \cdot |X_i|$ amongst all vertex-subsets $X_1, X_2, \ldots, X_s$ such that $\varphi(X_1, X_2, \ldots, X_s)$ is true.*

Theorem 3 should not be considered as completely new. As it was stated above, it was first proved by Courcelle and Vanicat [17], but under more restricted conditions. Still, their proof also applies to this more general case. It could also be deduced from the heavy machinery from [14]. Our presentation marginally differs from these previous works, while it avoids using explicitly some logic concepts such as MSO transductions. Roughly, we rewrite a $CMSO_1$ formula on a c-labelled graph as a longer $CMSO$ formula on the parse tree of its clique-width expression. Then, we make this parse tree of logarithmic depth, using a modified centroid decomposition, updating the $CMSO$ formula along the way. We end up designing a dynamic programming procedure on this parse tree, using prior work of Doner [24] and Thatcher and Wright [67] on tree automata.

4 Algorithms: the general case

Our main result in this section (Theorem 1) is proved in Sec. 4.3. In Sec. 4.1 we first compute the matching number, a key step toward the final proof of Theorem 1. Sec. 4.2 is devoted to computing the Edmonds-Gallai decomposition, and it is a gentle introduction to the techniques we also use in Sec. 4.3.

4.1 Size of a maximum matching

We explain in this section how to compute the matching number of bounded clique-width graphs. For that, we need a classic result from Matching theory:

▶ **Lemma 5** (Tutte-Berge formula [6, 7]). *For any graph $G = (V, E)$, we have:*

$$\nu(G) = \min_{U \subseteq V} \frac{1}{2}\left(|V| + |U| - odd(G \setminus U)\right)$$

where $odd(G \setminus U)$ denotes the number of connected components of odd size of $G \setminus U$.

Our main insight below is that evaluating the Tutte-Berge formula can be written as a $CMSO_1$ optimization problem. We prove it next:

▶ **Theorem 6.** *For any graph $G = (V, E)$ of clique-width at most k, if a k-expression is given then, we can compute $\nu(G)$ in $\tilde{\mathcal{O}}_k(n + m)$ time.*

Proof. By Theorem 4, it suffices to prove that the Tutte-Berge formula (see Lemma 5) can be written as a $CMSO_1$ optimization problem. For that, let us first define $inc(x, y, U) := inc(x, y) \wedge x \notin U \wedge y \notin U$ in order to suppress all edges incident to a given set U. The following formula can be used to test whether two vertices are in the same connected component of $G \setminus U$, for a given set U: $connected(x, y, U) := \forall X((x \in X \wedge y \notin X) \implies \exists x', y'(x' \in X \wedge y' \notin X \wedge inc(x', y', U)))$. Then, we can relate a vertex to its connected component of $G \setminus U$ as follows: $comp(x, X, U) := \forall y(y \in X \iff connected(x, y, U))$. We are now ready to define our formula for computing $\nu(G)$. It has two free variables.

$$\begin{aligned} TutteBerge(U, W) :=& \forall x \in W(x \notin U \wedge \exists X(comp(x, X, U) \wedge Card_{1,2}(X))) \\ & \wedge \forall x, y \in W(x = y \vee \neg connected(x, y, U)). \end{aligned}$$

The first line ensures that every vertex of W is in an odd component of $G \setminus U$. The second line ensures that two distinct vertices of W are in different components of $G \setminus U$. If we set $a_1 = 1, a_2 = -1$, the objective becomes to minimize $|U| - |W|$. Therefore, we get as solution $\delta = \min_{U \subseteq V}(|U| - odd(G \setminus U))$. By Lemma 5, we have $\nu(G) = \frac{1}{2}(n + \delta)$. ◀

This above Theorem 6 is the cornerstone of all the remainder of Sec. 4.

4.2 Edmonds-Gallai decomposition

We continue with a known structural result about maximum matchings in a graph. Recall that a graph is hypomatchable if the removal of any one vertex results in a graph with a perfect matching.

▶ **Theorem 7** (Edmonds-Gallai [30, 42, 43]). *Let $G = (V, E)$ be a graph, and let $A \subseteq V$ be the set of all vertices v so that there is a maximum matching of G that does not cover v. Set $B = N_G(A)$ and $C = V \setminus (A \cup B)$. Then:*

- *Every odd component H of $G \setminus B$ is hypomatchable and it has $V(H) \subseteq A$;*
- *Every even component H of $G \setminus B$ has a perfect matching and it has $V(H) \subseteq C$;*
- *For every $X \subseteq B$, the set $N(X)$ contains vertices in $> |X|$ odd components of $G \setminus B$.*

The partition (A, B, C) is called the Edmonds-Gallai decomposition *of G.*

In [10], the author proposes a randomized $\mathcal{O}(n^\omega)$-time algorithm for computing the Edmonds-Gallai decomposition of an n-vertex graph, where $\omega < 2.37286$ [2] denotes the exponent of square matrix multiplication. We improve this result to deterministic almost linear-time for all classes of bounded clique-width graphs (under the standard assumption in the field that a corresponding clique-width expression is given in the input):

▶ **Theorem 8.** *For any graph $G = (V, E)$ of clique-width at most k, if a k-expression is given then, we can compute its Edmonds-Gallai decomposition in $\tilde{\mathcal{O}}_k(n + m)$ time.*

Proof. If we are given the set A of all vertices left exposed by at least one maximum matching then, by Theorem 7, the sets B and C can be computed in additional $\mathcal{O}(n + m)$ time. Recall (see Theorem 6) that there exists a $CMSO_1$ formula $TutteBerge(U, W)$ to express that all vertices of W are in pairwise different odd components of $G \setminus U$. Let $EdmondsGallai(X, U, W) := TutteBerge(U \cup X, W)$. It is also a $CMSO_1$ formula since the union of two subsets can be easily expressed in MSO [14]. Then, for any X, let $\psi(X)$ be the problem of minimizing $|U| - |W|$ among all the subsets U, W such that $EdmondsGallai(X, U, W)$ is true. Observe that ψ is a $CMSO_1$ optimization function. We apply Theorem 3 to ψ. Then, we claim that $v \in A$ if and only if $\psi(\{v\}) = 2\nu(G) + 1 - n$. Indeed, by construction we have $\psi(\{v\}) = \min_{U \subseteq V \setminus \{v\}}(|U| - odd(G \setminus (\{v\} \cup U)))$, and therefore by Lemma 5, $\nu(G \setminus \{v\}) = \frac{1}{2}(n - 1 + \psi(\{v\}))$. Then:

$$\begin{aligned} v \in A \iff & \nu(G) = \nu(G \setminus \{v\}) \iff \nu(G) = \frac{1}{2}(n - 1 + \psi(\{v\})) \\ \iff & 2\nu(G) = n - 1 + \psi(\{v\}) \iff \psi(\{v\}) = 2\nu(G) + 1 - n. \end{aligned}$$

Computing $\nu(G)$ can be done in $\tilde{\mathcal{O}}_k(n + m)$ time (Theorem 6). Computing $\psi(\{v\})$ takes $\tilde{\mathcal{O}}_k(1)$ time per vertex v up to an $\tilde{\mathcal{O}}_k(n+m)$-time pre-processing (Theorem 3). As a result, we can compute the set A, and so, the Edmonds-Gallai decomposition, in $\tilde{\mathcal{O}}_k(n + m)$ time. ◀

4.3 Computation of a maximum matching

Let us first recall our main result in this section:

▶ **Theorem 1.** *Given a graph G and a corresponding k-expression, one can compute a maximum matching for G in deterministic $\tilde{\mathcal{O}}_k(n + m)$ time.*

Let us sketch our strategy to prove this above result. Given a graph $G = (V, E)$, we first recall that an edge-cut is, for a given subset A, the set of all edges between A and $V \setminus A$. It is balanced if we further have $\max\{|A|, |V \setminus A|\} \leq 2n/3$. Roughly, we compute a balanced edge-cut for G, we compute a subset of edges of the cut to be included in some maximum matching of G, then we recurse on subgraphs of $G[A]$ and $G[V \setminus A]$ separately.

The computation of a balanced edge-cut in $\tilde{\mathcal{O}}(k \cdot (n + m))$ time follows from prior works [19, 27] and is omitted here due to lack of space. An important property for this cut is that it can be edge-partitioned into at most k joins. We handle each join separately. For that, both Lemma 9 and Lemma 10 below apply Theorem 3 (distributed Courcelle's theorem).

▶ **Lemma 9.** *Let X, Y be the two sides of a join in a graph $G = (V, E)$, where $|X| \leq |Y|$ and $cw(G) \leq k$. If a k-expression is given then, in $\tilde{\mathcal{O}}_k(n+m)$ time, we can compute an inclusion-wise minimal subset $X' \subseteq X$ such that, in some maximum matching of G:*

1. *every vertex of X' is matched to some vertex of Y;*
2. *no vertex of $X \setminus X'$ is matched to a vertex of Y.*

▶ **Lemma 10.** *Let X, Y be the two sides of a join in a graph $G = (V, E)$, where $|X| \leq |Y|$ and $cw(G) \leq k$. We are given a subset $X' \subseteq X$ as stated in Lemma 9. If a k-expression is given then, in $\tilde{\mathcal{O}}_k(n+m)$ time, we can compute the intersection of the edges of the join with some maximum matching of G.*

Finally, once we computed from a join a subset of edges to be added into a maximum matching, all other edges of the join can be removed from the graph. We must also remove all the end-vertices of the edges included into the matching. Doing so, we need the following two lemmas in order to update the k-expression of the graph considered.

▶ **Lemma 11.** *Let $G = (V, E)$ be a graph, let (U, W) be a cut of G, and let $U' \subseteq U$. If $cw(G) \leq k$ then the graph H, obtained from G by removing all edges between U' and $W' := N(U') \cap W$, has clique-width at most $3k$. Furthermore, we can compute a $3k$-expression of H from a k-expression of G in $\mathcal{O}(n+m)$ time.*

▶ **Lemma 12.** *Let $G = (V, E)$ be a graph and let $H = (X, E_X)$ be an induced subgraph of G. If a k-expression of G is given then, in $\mathcal{O}(k \cdot (n+m))$ time, we can compute a k-expression of H of size $\mathcal{O}(|X| + |E_X|)$.*

We are now ready to prove the main result in this section:

Sketch Proof of Theorem 1. We compute some balanced cut (U, W) and a partition $U_1, U_2, \ldots, U_k$ of U such that the edges of the cut are partitioned into k joins (one of them having possibly no edge) with respective sides U_i and $W_i = N(U_i) \cap W$ for every i. Details are omitted due to lack of space. Since we are given the U_i's, all the corresponding sides W_i can be computed in total $\mathcal{O}(n+m)$ time. We consider these k joins sequentially, from $i = 1$ to $i = k$. At each step i, we are given a subgraph G_i of G such that $cw(G_i) \leq 3k$ and a corresponding $3k$-expression is given (initially, $G_1 := G$). We apply Lemmas 9 and 10 in order to compute the intersection of a maximum matching of G_i with the join with sides U_i, W_i. It takes $\tilde{\mathcal{O}}_k(n+m)$ time. Denote $F_i \subseteq U_i \times W_i$ the set of edges in this intersection, and let $V(F_i)$ be the set of vertices incident to an edge of F_i. We obtain G_{i+1} from G_i by removing the vertices in $V(F_i)$ and all remaining edges between $U_i \setminus V(F_i)$ and $W_i \setminus V(F_i)$.

Let $M_i := \bigcup_{j=1}^{i} F_j$ be the matching constructed so far, and let $V(M_i)$ be the set of vertices incident to it. Let. also $X_i := \bigcup_{j=1}^{i} U_j$. By induction, the union of M_i with a maximum matching of G_{i+1} is a maximum matching of G. Also by induction, G_{i+1} is obtained from $G \setminus V(M_i)$ by removing all edges between $X_i \setminus V(M_i)$ and $W \setminus V(M_i)$. Therefore, we can apply Lemma 12 (for $X = V \setminus V(M_i)$) then Lemma 11 (for $U' = X_i \setminus V(M_i)$) to compute a $3k$-expression of G_{i+1}. It takes $\mathcal{O}(k \cdot (n+m))$ time. – Note that since we always apply Lemma 11 to G, and not to the G_i's, there is no blow-up of the clique-width value, *i.e.*, the clique-width of any G_i is at most $3k$. –

Since all k joins got removed, we are left with computing a maximum matching in $G[U \setminus V(M_k)]$ and in $G[W \setminus V(M_k)]$ respectively. Apply Lemma 12 to compute k-expressions for both subgraphs, then call our above algorithm recursively in order to compute a maximum matching. Since the cut is balanced, the recursive depth is in $\mathcal{O}(\log n)$. ◀

5 Algorithms: the bipartite case

We present in this section an alternative to Theorem 1, with polynomial dependency on the clique-width, but only for bipartite graphs (see Theorem 2). The structure of bipartite graphs of bounded clique-width can be quite complex. For instance, let $G = (V, E)$ be any graph and let $V' = \{v' \mid v \in V\}$ be a disjoint copy of V. We can define the bipartite graph $B_G = (V \cup V', \{u'v \mid uv \in E\})$ and, if G has clique-width at most k, then B_G has clique-width at most $2k$. Some classes of monogenic bipartite graphs are also known to have bounded clique-width [21].

5.1 Reduction to Maximum b-matching

Let $G = (V, E)$ be a graph and let $b : V \to \mathbb{N}$ assign a non negative capacity to each vertex. We say that $x : E \to \mathbb{N}$ is a b-matching if we have $\sum_{u \in N_G(v)} x_{uv} \leq b(v)$ for each vertex v. Observe that a matching is a b-matching for the trivial function $b(v) = 1$ for each $v \in V$. The cardinality of a b-matching is defined as $\|x\|_1 = \sum_{e \in E} x_e$. We denote by $\nu(G, b)$ the cardinality of a maximum b-matching in G. Let also $\|b\|_1 = \sum_{v \in V} b(v)$ be the sum of all the vertex capacities. More generally, for every vertex-subset S, let $b(S) = \sum_{v \in S} b(v)$. Given a b-matching x, and a vertex v, let also $d_x(v) := \sum_{u \in N_G(v)} x_{uv} \leq b(v)$.

We start with the following reduction rule:

▶ **Lemma 13.** *Let (G, b) be some instance of* Maximum b-Matching*, and let U and W be disjoint vertex-subsets such that there is a join between U and W. Consider the new instance (G', b') obtained from (G, b) by removing all edges between U and W, adding two new vertices $u, w \notin V(G)$ and edges $\{uw\} \cup \{uv \mid v \in U\} \cup \{wv' \mid v' \in W\}$, and finally setting $b'(u) = b'(w) = \min\{b(U), b(W)\}$. Then, we have $\nu(G', b') = \nu(G, b) + \min\{b(U), b(W)\}$.*

Moreover, if G is bipartite, then so is G'.

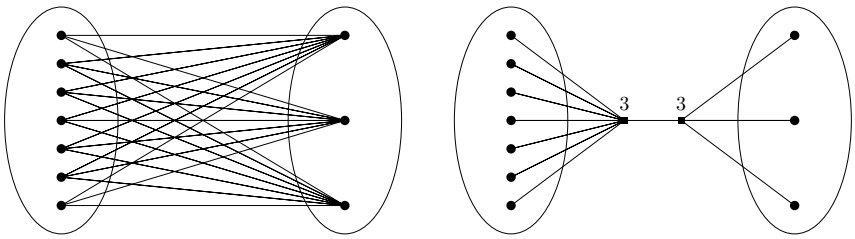


Figure 2 Transformation of Lemma 13.

By repeatedly applying Lemma 13 to some special balanced edge-cuts, we obtain:

▶ **Proposition 14.** *There is an $\mathcal{O}(k \cdot (n+m) \log n)$-time reduction from* Maximum Matching *on graphs with clique-width at most k (if a k-expression is known) to* Maximum b-Matching *on graphs with tree-width $\mathcal{O}(k \log n)$. For the resulting instance (H, b), the algorithm also outputs a corresponding tree decomposition. Furthermore, $|V(H)| \leq \|b\|_1 \leq \mathcal{O}(\min\{n + m, n \log n\})$, and if G is bipartite, then so is H.*

5.2 Reduction to Linear Programming

The Maximum b-Matching problem is a classic example of an integer linear program. It is well-known that for the special case of Maximum Matching within *bipartite* graphs, we can drop the condition for all variables to be integers, thus reducing the computation of the

matching number to the solving of a linear program [55]. Because of Tutte's quasi-polynomial reduction from Maximum b-Matching to Maximum Matching [69], this is also true for Maximum b-Matching within bipartite graphs. Very recently, Dong et. al. [25] answered an open question from Fomin et. al. [36] about bounded tree-width linear programs. We restate below their main result:

▶ **Theorem 15** ([25]). *Given a linear program* $\max_{Mx=b,\ell\leq x\leq u} c^\intercal x$ [4]*, where* $M \in \mathbb{R}^{d\times n}$ *is a full-rank matrix with* $d \leq n$*, define the* **dual graph** G_M *to be the graph with vertex-set* $\{1, 2\ldots, d\}$*, such that* $ij \in E(G_M)$ *if there is a column* r *such that* $M_{i,r} \neq 0$ *and* $M_{j,r} \neq 0$*. Suppose that a tree decomposition of* G_M *with width* τ *is given, and* R *is the diameter of the polytope, namely, for any* $\ell \leq x \leq u$ *with* $Mx = b$*, we have* $\|x\|_2 \leq R$*. Then, for any* $0 < \varepsilon \leq 1$*, we can find* $\ell \leq x \leq u$ *such that*

$$c^\intercal x \geq \max_{Mx=b,\ell\leq x\leq u} c^\intercal x - \varepsilon \cdot \|c\|_2 \cdot R \text{ and } \|Mx - b\|_2 \leq \varepsilon \cdot (\|M\|_2 \cdot R + \|b\|_2)$$

in expected time $\tilde{\mathcal{O}}(n \cdot \tau^2 \log(1/\varepsilon))$.

Dong et. al. [25] referred to [66] and [53] for a detailed discussion about converting an approximate solution to an exact solution. We give a direct proof for Maximum b-Matching within bipartite graphs. For that, we combine a folklore reduction to Maximum Flow with a nice rounding technique by Madry [56].

▶ **Proposition 16.** *The* Maximum b-Matching *problem within bipartite graphs of tree-width at most* τ *can be solved in expected* $\tilde{\mathcal{O}}(n\tau^2)$ *time, if a corresponding tree decomposition is given in the input.*

Proof. Let $G = (V_0 \cup V_1, E)$ be a bipartite graph and let $b : V_0 \cup V_1 \to \mathbb{N}$. The incidence matrix of G is the $n \times m$ matrix M such that $M_{v,e} = 1$ if v is an end-vertex of edge e and $M_{v,e} = 0$ otherwise. Let also c be the all-one vector. To compute the cardinality of a maximum b-matching for G, it suffices to solve the linear program $\max_{Mx\leq b} c^\intercal x$. We slightly modify this above program so that we fit in the conditions of Theorem 15. First, if we let $\ell, u \in \mathbb{R}^m$ such that ℓ is all-zero and $u_{vv'} = \min\{b(v), b(v')\}$ for each edge $vv' \in E$, then we must now solve $\max_{Mx\leq b,\ell\leq x\leq u} c^\intercal x$. If we further add n new variables $(x_v)_{v\in V}$ such that $0 \leq x_v \leq b(v)$ for each $v \in V$, then we can replace all constraints by $x_v + \sum_{vv'\in E} x_{vv'} = b(v)$ for each vertex $v \in V$, thus getting a new linear program $\max_{M'x=b,\ell'\leq x\leq u'} c'^\intercal x$ to solve, where $M' \in \mathbb{R}^{n\times(m+n)}$.

Note that, since we constructed M' from M by adding n new columns with exactly one nonzero value each, we have $G_{M'} = G_M = G$. Furthermore, an easy upper bound on the diameter R of the polytope is $R \leq \|b\|_2 \leq \|b\|_1$. We also have $\|c'\|_2 = \sqrt{m}$ and $\|M'\|_2 \leq \sqrt{\sum_v (1 + d(v))^2} \leq n^{3/2}$.

Assume G to be given with a tree decomposition of width at most τ, and apply Theorem 15 to the above linear program with $\varepsilon = 1/(4 \cdot \|b\|_1 \cdot n^2)$. We denote by x its output. Set all variables $x_v, v \in V$ to 0. Then, for all vertices v such that $\sum_{vv'\in E} x_{v'v} > b(v)$, we decrease the variables $x_{v'v}$ of incident edges until we reach equality. In doing so, we obtain a fractional b-matching y. By construction:

$$\begin{aligned}\|x\|_1 - \|y\|_1 &\leq \sum_v \max\{0, (M'x)_v - b(v)\} \leq \|M'x - b\|_1 \leq \sqrt{n} \cdot \|M'x - b\|_2 \\ &\leq \varepsilon\sqrt{n} \cdot (\|M'\|_2 \cdot R + \|b\|_2) \leq \varepsilon\sqrt{n} \cdot \|b\|_1 \cdot (n^{3/2} + 1) \leq 2\varepsilon n^2 \cdot \|b\|_1 \leq 1/2.\end{aligned}$$

[4] The result is stated in [25] for minimization problems. Since we only consider it here for Maximum b-Matching, we rather write it as a maximization problem.

We now construct a network D from G by adding two new vertices s and t, an arc (s,u) for every $u \in V_0$, an arc (v,t) for every $v \in V_1$, and finally by orienting all the edges of G from V_0 to V_1. The capacities of the arcs are defined as follows: $\kappa(s,v) = b(v)$ for every $v \in V_0$; $\kappa(v',t) = b(v')$ for every $v' \in V_1$; $\kappa(v,v') = \min\{b(v), b(v')\}$ for every $vv' \in E$ such that $v \in V_0, v' \in V_1$. Then, we construct a fractional st-flow as follows: $f_y(s,v) = \sum_{vv' \in E} y_{vv'}$ for every $v \in V_0$; $f_y(v',t) = \sum_{vv' \in E} y_{vv'}$ for every $v' \in V_1$; and $f_y(v,v') = y_{vv'}$ for every $vv' \in E$ such that $v \in V_0, v' \in V_1$. Note that the value of this flow is exactly $\|y\|_1$. Let f'_y be an integral st-flow of value $\lfloor \|y\|_1 \rfloor$. It can be constructed from f_y in $\tilde{\mathcal{O}}(m) = \tilde{\mathcal{O}}(\tau n)$ time [56, Corollary 3.4]. There is a one-to-one mapping between b-matchings in G and integral st-flows in D. In particular, the maximum value of a st-flow is exactly $\nu(G,b)$. Furthermore,

$$\|y\|_1 \geq \|x\|_1 - 1/2 \geq \nu(G,b) - \varepsilon \cdot \|c\|_2 \cdot R - 1/2 \geq \nu(G,b) - 1.$$

Therefore, we can transform f'_y into a maximum st-flow f_z by computing at most one augmenting path in the residual graph $D_{f'_y}$. It can be done in $\mathcal{O}(m) = \mathcal{O}(\tau n)$ time. The resulting b-matching is maximum: for every $vv' \in E$ with $v \in v_0, v' \in V_1$, we set $z_{vv'} = f_z(v,v')$. ◀

We are finally ready to prove our second main result in this paper:

▶ **Theorem 2.** *Given a bipartite graph G and a corresponding k-expression, one can compute a maximum matching for G in randomized $\tilde{\mathcal{O}}(k^2 \cdot (n+m))$ time.*

Proof. The result follows from the combination of Proposition 14 with Proposition 16. ◀

6 Conclusion

Our first main result in the paper is a quasi linear-time algorithm for computing a maximum matching within the graphs of bounded clique-width. We left open whether an $\tilde{\mathcal{O}}(k^c \cdot (n+m))$-time algorithm exists within the graphs of clique-width at most k, for some constant c. Our second main result is to prove the existence of such an algorithm for the bipartite graphs of clique-width at most k. For that, we reduce the Maximum Matching problem on bounded clique-width (bipartite) graphs to the Maximum b-Matching problem on bounded tree-width (bipartite) graphs. We left open the complexity of Maximum b-Matching within bounded tree-width graphs in general. Furthermore, we observe that an $\tilde{\mathcal{O}}(k^c \cdot (n+m))$-time algorithm for this problem within the graphs of tree-width at most k would imply a similar algorithm for Maximum Matching within graphs of clique-width at most k.

References

1 Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 377–391. SIAM, 2016. doi:10.1137/1.9781611974331.ch28.

2 J. Alman and V. Vassilevska Williams. A refined laser method and faster matrix multiplication. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. SIAM, 2021.

3 N. Anari and V. Vazirani. Matching Is as Easy as the Decision Problem, in the NC Model. In *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*, volume 151 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 54:1–54:25. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2020.

4 R.P. Anstee. A polynomial algorithm for b-matchings: an alternative approach. *Information Processing Letters*, 24(3):153–157, 1987.

5 M. Bentert, T. Fluschnik, A. Nichterlein, and R. Niedermeier. Parameterized aspects of triangle enumeration. *Journal of Computer and System Sciences*, 103:61–77, 2019.

6 C. Berge. Sur le couplage maximum dun graphe. *COMPTES RENDUS HEBDOMADAIRES DES SEANCES DE L'ACADEMIE DES SCIENCES*, 247(3):258–259, 1958.

7 John Adrian Bondy and Uppaluri Siva Ramachandra Murty. *Graph theory*, volume 244 of *Graduate Texts in Mathematics*. Springer-Verlag London, 2008.

8 M.L. Carmosino, J. Gao, R. Impagliazzo, I. Mihajlin, R. Paturi, and S. Schneider. Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, pages 261–270, 2016.

9 M. Chang. Algorithms for maximum matching and minimum fill-in on chordal bipartite graphs. In *ISAAC*, pages 146–155. Springer, 1996.

10 J. Cheriyan. Randomized $\tilde{\mathcal{O}}(M(|V|))$ Algorithms for Problems in Matching Theory. *SIAM Journal on Computing*, 26(6):1635–1655, 1997.

11 V. Cohen-Addad, M. Habib, and F. de Montgolfier. Algorithmic aspects of switch cographs. *Discrete Applied Mathematics*, 200:23–42, 2016.

12 Derek G. Corneil and Udi Rotics. On the relationship between clique-width and treewidth. *SIAM Journal on Computing*, 34(4):825–847, 2005. doi:10.1137/S0097539701385351.

13 D. Coudert, G. Ducoffe, and A. Popa. Fully polynomial FPT algorithms for some classes of bounded clique-width graphs. *ACM Transactions on Algorithms (TALG)*, 15(3):1–57, 2019.

14 B. Courcelle and J. Engelfriet. *Graph structure and monadic second-order logic: a language-theoretic approach*, volume 138. Cambridge University Press, 2012.

15 B. Courcelle, J.A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33(2):125–150, 2000.

16 B. Courcelle and A. Twigg. Constrained-path labellings on graphs of bounded clique-width. *Theory of Computing Systems*, 47(2):531–567, 2010.

17 B. Courcelle and R. Vanicat. Query efficient implementation of graphs of bounded clique-width. *Discrete Applied Mathematics*, 131(1):129–150, 2003.

18 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

19 Bruno Courcelle, Pinar Heggernes, Daniel Meister, Charis Papadopoulos, and Udi Rotics. A characterisation of clique-width through nested partitions. *Discrete Applied Mathematics*, 187:70–81, 2015. doi:10.1016/j.dam.2015.02.016.

20 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1):77–114, 2000. doi:10.1016/S0166-218X(99)00184-5.

21 K.K. Dabrowski and D. Paulusma. Classifying the clique-width of H-free bipartite graphs. *Discrete Applied Mathematics*, 200:43–51, 2016.

22 E. Dahlhaus and M. Karpinski. Matching and multidimensional matching in chordal and strongly chordal graphs. *Discrete Applied Mathematics*, 84(1-3):79–91, 1998.

23 Reinhard Diestel. *Graph Theory*. Graduate Texts in Mathematics. Springer, 2010. 4*th* edition. doi:10.1007/978-3-662-53622-3.

24 J. Doner. Tree acceptors and some of their applications. *Journal of Computer and System Sciences*, 4(5):406–451, 1970.

25 S. Dong, Y.T. Lee, and G. Ye. A Nearly-Linear Time Algorithm for Linear Programs with Small Treewidth: A Multiscale Representation of Robust Central Path. In *ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1784–1797, 2021.

26 F. Dragan. On greedy matching ordering and greedy matchable graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, volume 1335 of *LNCS*, pages 184–198. Springer, 1997.

27 G. Ducoffe. Optimal diameter computation within bounded clique-width graphs. Technical Report 2011.08448, arXiv, 2020.

28 G. Ducoffe and A. Popa. The b-Matching Problem in Distance-Hereditary Graphs and Beyond. In *29th International Symposium on Algorithms and Computation (ISAAC 2018)*, volume 123 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:13. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018.

29 G. Ducoffe and A. Popa. The use of a pruned modular decomposition for Maximum Matching algorithms on some graph classes. *Discrete Applied Mathematics*, 291:201–222, 2021.

30 J. Edmonds. Paths, trees, and flowers. *Canadian J. of mathematics*, 17(3):449–467, 1965.

31 Wolfgang Espelage, Frank Gurski, and Egon Wanke. How to solve NP-hard graph problems on clique-width bounded graphs in polynomial time. In *International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, volume 1 of *LNCS*, pages 117–128. Springer, 2001. doi:10.1007/3-540-45477-2_12.

32 T. Fluschnik, G.B. Mertzios, and A. Nichterlein. Kernelization lower bounds for finding constant-size subgraphs. In *Conference on Computability in Europe*, pages 183–193. Springer, 2018.

33 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of clique-width parameterizations. *SIAM Journal on Computing*, 39(5):1941–1956, 2010. doi:10.1137/080742270.

34 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Almost optimal lower bounds for problems parameterized by clique-width. *SIAM Journal on Computing*, 43(5):1541–1563, 2014. doi:10.1137/130910932.

35 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Clique-width III: Hamiltonian Cycle and the Odd Case of Graph Coloring. *ACM Transactions on Algorithms (TALG)*, 15(1):9, 2019. doi:10.1145/3280824.

36 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, Michal Pilipczuk, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. *ACM Transactions on Algorithms (TALG)*, 14(3):34:1–34:45, 2018. doi:10.1145/3186898.

37 J.-L. Fouquet, V. Giakoumakis, and J.-M. Vanherpe. Bipartite graphs totally decomposable by canonical decomposition. *International J. of Foundations of Computer Science*, 10(04):513–533, 1999.

38 J.-L. Fouquet, I. Parfenoff, and H. Thuillier. An $O(n)$-time algorithm for maximum matching in P_4-tidy graphs. *Information processing letters*, 62(6):281–287, 1997.

39 Martin Fürer. A natural generalization of bounded tree-width and bounded clique-width. In *Latin American Symposium on Theoretical Informatics*, pages 72–83. Springer, 2014.

40 H.N. Gabow. Data structures for weighted matching and extensions to b-matching and f-factors. *ACM Transactions on Algorithms (TALG)*, 14(3):1–80, 2018.

41 H.N. Gabow and P. Sankowski. Algorithms for Weighted Matching Generalizations I: Bipartite Graphs, b-matching, and Unweighted f-factors. *SIAM Journal on Computing*, 50(2):440–486, 2021.

42 T. Gallai. Kritische graphen II. *Magyar Tud. Akad. Mat. Kutato Int. Kozl.*, 8:373–395, 1963.

43 T. Gallai. Maximale systeme unabhangiger kanten. *Magyar Tud. Akad. Mat. Kutato Int. Kozl.*, 9:401–413, 1964.

44 A. Gerards. Matching. *Handbooks in operations research and management science*, 7:135–224, 1995.

45 Archontia C. Giannopoulou, George B. Mertzios, and Rolf Niedermeier. Polynomial fixed-parameter algorithms: A case study for longest path on interval graphs. *Theoretical Computer Science*, 689:67–95, 2017. doi:10.1016/j.tcs.2017.05.017.

46 F. Glover. Maximum matching in a convex bipartite graph. *Naval Research Logistics (NRL)*, 14(3):313–316, 1967.

47 Martin Charles Golumbic and Udi Rotics. On the clique-width of some perfect graph classes. *International Journal of Foundations of Computer Science*, 11(03):423–443, 2000. doi:10.1142/S0129054100000260.

48 Torben Hagerup, Jyrki Katajainen, Naomi Nishimura, and Prabhakar Ragde. Characterizing multiterminal flow networks and computing flows in networks of small treewidth. *Journal of Computer and System Sciences*, 57(3):366–375, 1998.

49 F. Hegerfeld and S. Kratsch. On Adaptive Algorithms for Maximum Matching. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132, pages 71:1–71:16. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019.

50 Yoichi Iwata, Tomoaki Ogasawara, and Naoto Ohsaka. On the power of tree-depth for fully polynomial FPT algorithms. In *International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 96 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 41:1–41:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.STACS.2018.41.

51 S. Kratsch and F. Nelles. Efficient and Adaptive Parameterized Algorithms on Modular Decompositions. In *European Symposium on Algorithms (ESA)*, pages 55:1–55:15, 2018. doi:10.4230/LIPIcs.ESA.2018.55.

52 S. Kratsch and F. Nelles. Efficient Parameterized Algorithms for Computing All-Pairs Shortest Paths. In *International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 154, pages 38:1–38:15. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020.

53 Y. T. Lee and A. Sidford. Path Finding I: Solving Linear Programs with\~ O (sqrt (rank)) Linear System Solves. Technical Report 1312.6677, arXiv, 2013.

54 Y. Liang and C. Rhee. Finding a maximum matching in a circular-arc graph. *Information processing letters*, 45(4):185–190, 1993.

55 L. Lovász and M. Plummer. *Matching theory*, volume 367. American Mathematical Soc., 2009.

56 A. Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 253–262. IEEE, 2013.

57 Johann A. Makowsky and Udi Rotics. On the clique-width of graphs with few P_4's. *International Journal of Foundations of Computer Science*, 10(03):329–348, 1999. doi:10.1142/S0129054199000241.

58 G. Mertzios, A. Nichterlein, and R. Niedermeier. A Linear-Time Algorithm for Maximum-Cardinality Matching on Cocomparability Graphs. *SIAM Journal on Discrete Mathematics*, 32(4):2820–2835, 2018.

59 George B. Mertzios, André Nichterlein, and Rolf Niedermeier. The power of linear-time data reduction for maximum matching. In *International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 83 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 46:1–46:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.MFCS.2017.46.

60 S. Micali and V. Vazirani. An $O(\sqrt{V}E)$ algorithm for finding maximum matching in general graphs. In *FOCS'80*, pages 17–27. IEEE, 1980.

61 A. Moitra and R. Johnson. A parallel algorithm for maximum matching on interval graphs. In *ICPP*, 1989.

62 S. Oum and P. Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B*, 96(4):514–528, 2006.

63 M.W. Padberg and M.R. Rao. The Russian method for linear inequalities III: Bounded integer programming. Technical Report 78, INRIA, 1981.

64 W.R. Pulleyblank. *Faces of Matching Polyhedra*. PhD thesis, Univ. of Waterloo, Dept. Combinatorics and Optimization, 1973.

65 Michaël Rao. Solving some NP-complete problems using split decomposition. *Discrete Applied Mathematics*, 156(14):2768–2780, 2008. doi:10.1016/j.dam.2007.11.013.

66 J. Renegar. A polynomial-time algorithm, based on Newton's method, for linear programming. *Mathematical programming*, 40(1):59–93, 1988.

67 J.W. Thatcher and J.B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical systems theory*, 2(1):57–81, 1968.

68 N. Trinajstić, D.J. Klein, and M. Randić. On some solved and unsolved problems of chemical graph theory. *International Journal of Quantum Chemistry*, 30(S20):699–742, 1986.

69 W. Tutte. A short proof of the factor theorem for finite graphs. *Canad. J. Math*, 6(1954):347–352, 1954.

70 W.T. Tutte. The factorization of linear graphs. *Journal of the London Mathematical Society*, 1(2):107–111, 1947.

71 W.T. Tutte. Antisymmetrical digraphs. *Canadian Journal of Mathematics*, 19:1101–1117, 1967.

72 V. Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the ICM*, volume 3, pages 3431–3472. World Scientific, 2018.

73 V. Vassilevska Williams and R.R. Williams. Subcubic equivalences between path, matrix, and triangle problems. *Journal of the ACM (JACM)*, 65(5):1–38, 2018.

74 M.-S. Yu and C.-H. Yang. An $O(n)$-time algorithm for maximum matching on cographs. *Information processing letters*, 47(2):89–93, 1993.

75 R. Yuster. Maximum matching in regular and almost regular graphs. *Algorithmica*, 66(1):87–92, 2013.

76 R. Yuster and U. Zwick. Maximum matching in graphs with an excluded minor. In *Proceedings of the eighteenth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 108–117. Society for Industrial and Applied Mathematics, 2007.

Optimal Centrality Computations Within Bounded Clique-Width Graphs

Guillaume Ducoffe
National Institute for Research and Development in Informatics, Bucharest, Romania
University of Bucharest, Romania

Abstract

Given an n-vertex m-edge graph G of clique-width at most k, and a corresponding k-expression, we present algorithms for computing some well-known centrality indices (eccentricity and closeness) that run in $\mathcal{O}(2^{\mathcal{O}(k)}(n+m)^{1+\epsilon})$ time for any $\epsilon > 0$. Doing so, we can solve various distance problems within the same amount of time, including: the diameter, the center, the Wiener index and the median set. Our run-times match conditional lower bounds of Coudert et al. (*SODA'18*) under the Strong Exponential-Time Hypothesis. On our way, we get a distance-labeling scheme for n-vertex m-edge graphs of clique-width at most k, using $\mathcal{O}(k \log^2 n)$ bits per vertex and constructible in $\tilde{\mathcal{O}}(k(n+m))$ time from a given k-expression. Doing so, we match the label size obtained by Courcelle and Vanicat (*DAM 2016*), while we considerably improve the dependency on k in their scheme. As a corollary, we get an $\tilde{\mathcal{O}}(kn^2)$-time algorithm for computing All-Pairs Shortest-Paths on n-vertex graphs of clique-width at most k, being given a k-expression. This partially answers an open question of Kratsch and Nelles (*STACS'20*). Our algorithms work for graphs with non-negative vertex-weights, under two different types of distances studied in the literature. For that, we introduce a new type of orthogonal range query as a side contribution of this work, that might be of independent interest.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Fixed parameter tractability; Theory of computation → Shortest paths

Keywords and phrases Clique-width, Centralities computation, Facility Location problems, Distance-labeling scheme, Fine-grained complexity in P, Graph theory

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.16

Funding This work was supported by project PN-19-37-04-01 "New solutions for complex problems in current ICT research fields based on modelling and optimization", funded by the Romanian Core Program of the Ministry of Research and Innovation (MCI) 2019-2022.

1 Introduction

For any undefined graph terminology, see [2, 25]. Unless stated otherwise, all graphs considered in this work are simple and connected. We here consider *clique-width*, which is one of the most studied parameters in Graph Theory, superseded only by treewidth. Roughly, clique-width is a measure of the closeness of a graph to a cograph (*a.k.a.*, P_4-free graph). We postpone its formal definition until Sec. 2. The clique-width was shown to be bounded on many important subclasses of perfect graphs [4, 5, 22, 40, 48], and beyond [9, 6, 10, 7, 8, 23, 49, 53, 54]. For instance, distance-hereditary graphs, and so, trees, have clique-width at most three [40]. Every graph of bounded treewidth also has bounded clique-width, but the converse is not true [15]. Indeed, unlike for treewidth, there are dense graphs of bounded clique-width (*e.g.*, the complete graphs). This generality comes at some cost: whereas the celebrated Courcelle's theorem asserts that any problem expressible in MSO_2 logic can be solved in FPT linear time on bounded treewidth graphs [17], the same is true for bounded clique-width graphs only for the problems expressible in the more restricted MSO_1 logic [19]. Fomin et al. showed this to be unavoidable, in the sense that there are problems expressible in MSO_2 logic that are $W[1]$-hard in the clique-width [31, 32, 33]. We refer to [29] for other algorithmic applications of clique-width in parameterized complexity.

Our focus is about the so-called "FPT in P" program. Here the goal is, for some problem solvable in $\mathcal{O}(m^{q+o(1)})$ time on arbitrary m-edge graphs, to design an $\mathcal{O}(f(k)m^{p+o(1)})$-time algorithm, for some $p < q$, within the class of graphs where some fixed parameter is at most k (one usually seeks for $p = 1$ and $f(k) = k^{\mathcal{O}(1)}$). The idea of using tools and methods from parameterized complexity in order to solve faster certain polynomial-time solvable problems has been here and there in the literature for a while (*e.g.*, see [42]). Nevertheless it was only recently that such idea was better formalized [38], in part motivated by some surprising results obtained for treewidth [1]. Indeed, on the positive side, the treewidth does help in solving faster many important problems in P, that is, in $\tilde{\mathcal{O}}(k^{\mathcal{O}(1)}n)$ time on graphs and matrices of treewidth at most k [34, 44]. But for other such problems, any truly subquadratic-time parameterized algorithm requires *exponential* dependency on the treewidth. For example, given a graph G with a non-negative weight function on its edge-set (resp., on its vertex-set), the weight of a path equals the sum of the weights of all its edges (resp., of all its vertices). For unweighted graphs, this is exactly the number of edges (resp, the number of edges plus one). The distance $d_G(u, v)$ between two vertices u and v is equal to the least weight of a uv-path. Finally, the *diameter* of G is defined as $diam(G) = \max_{u,v \in V(G)} d_G(u, v)$. Abboud et al. proved that under the Strong Exponential-Time Hypothesis (SETH), for any $\epsilon > 0$, there is no $\mathcal{O}(2^{o(k)}n^{2-\epsilon})$-time algorithm for computing the diameter of n-vertex *unweighted* graphs of treewidth at most k [1]. An algorithm for this problem on *weighted* graphs, running in $\mathcal{O}(2^{\mathcal{O}(k)}n^{1+\epsilon})$ time for any $\epsilon > 0$, was proved recently in [11] by using the orthogonal range query framework of Cabello and Knauer [13].

Insofar, clique-width has received less attention than treewidth in the nascent field of FPT in P. Perhaps one good reason for that is that, for most problems on edge-weighted graphs, clique-width provably does *not* help [46]. This is because we may regard any graph as a weighted clique, where each non-edge got replaced by an edge of sufficiently large weight. Note however that most conditional lower bounds in the literature hold even for unweighted graphs (this is the case for the diameter and the other distance problems that we here study). Furthermore, in a recent paper Kratsch and Nelles [47] have evidenced that some applications of clique-width to unweighted graphs could be extended to vertex-weighted graphs. We give further evidence for that in our work. One other well-known drawback of clique-width is that, unlike for treewidth, the parameterized complexity of computing it is a wide open problem [14]. To date, the best-known approximation algorithms for clique-width run in $\mathcal{O}(n^3)$-time [50]. Still, on many subclasses of bounded clique-width graphs, there exist linear-time algorithms in order to compute a so called "k-expression", for some $k = \mathcal{O}(1)$, with the latter certifying the clique-width of the graph to be at most k [40, 49]. Therefore, the study of graph problems in P parameterized by clique-width may be regarded as a unifying framework for all such subclasses. In this respect, Coudert et al. obtained $\tilde{\mathcal{O}}(k^{\mathcal{O}(1)}(n + m))$-time algorithms for triangle and cycle problems on n-vertex m-edge graphs of clique-width at most k [16]. However, they also observed that assuming SETH, even on n-vertex cubic graphs of clique-width at most k, for any $\epsilon > 0$, there is no $\mathcal{O}(2^{o(k)}n^{2-\epsilon})$-time algorithm for computing the diameter. Unlike for treewidth, it was open until this paper whether there does exist a parameterized quasi-linear-time algorithm for this problem on bounded clique-width graphs that matches their conditional lower bound. Indeed, we are only aware of a linear-time algorithm for computing the diameter of bounded clique-width graphs in [20], but with a super-exponential dependency on the clique-width in the runtime, due to the use of Courcelle's theorem. The work of Coudert et al. has also been continued in [28, 27, 46] and especially in [47], where the authors obtained an $\mathcal{O}((kn)^2)$-time algorithm for All-Pairs Shortest Paths (APSP) on n-vertex graphs of clique-width at most k.

Results. We provide new insights on the fine-grained complexity of polynomial-time solvable distance problems within bounded clique-width graphs. As in all previous works in this area, all our algorithmic results require a k-expression to be given in the input. Specifically, let $G = (V, E, w)$ be such that $|V| = n$, $|E| = m$, and $w : V \to \mathbb{N}$. The *eccentricity* of a vertex u, denoted $e_G(u)$, is its largest distance to any other vertex; its inverse is sometimes called the graph centrality of u [41]. The *closeness centrality* of u, denoted $C_G(u)$, equals $1/\sum_v d_G(u, v)$ [52]. For a discussion about these centrality measures, and others, and their role in social network analysis, we refer to [24]. Our main contribution is an algorithm for computing all eccentricities, and closeness centralities within the n-vertex m-edge graphs of clique-width at most k, being given a k-expression, that runs in $\mathcal{O}(2^{\mathcal{O}(k)}(n+m)^{1+\epsilon})$ time for any $\epsilon > 0$ (Theorem 8).

We point out that the diameter of a graph is its largest eccentricity. The radius of a graph is its least eccentricity, and its center is the set of all vertices whose eccentricity equals the radius. Therefore, our result for computing all eccentricities implies, for any $\epsilon > 0$, an $\mathcal{O}(2^{\mathcal{O}(k)}(n+m)^{1+\epsilon})$-time algorithm for computing the diameter, the radius, and the center of a graph of clique-width at most k, if a k-expression is given. To the best of our knowledge, it is the first algorithm to match the conditional lower bound of Coudert et al. Previously, the only known algorithms for these problems were applications of Courcelle's theorem [19]. The Wiener index $W(G)$ of a graph G is the sum of all its distances, while its median set contains all the vertices of maximal closeness centrality. In the same way, our result for computing the closeness centrality implies, for any $\epsilon > 0$, an $\mathcal{O}(2^{\mathcal{O}(k)}(n+m)^{1+\epsilon})$-time algorithm for computing both the Wiener index and the median set of a graph of clique-width at most k, if a k-expression is given. Our runtimes are also optimal under SETH for the Wiener index, and so, for the closeness centrality (the conditional lower bound is the same as for the diameter problem, see the discussion in Sec. 4).

Recall that our results hold for vertex-weighted graphs. A related problem, studied in location theory, is given an unweighted graph $G = (V, E)$ and a cost function $p : V \to \mathbb{N}$, to compute for every vertex u its p-eccentricity (resp., its total p-distance sum), defined as $e_p(u) := \max_v p(v)d_G(u, v)$ (resp., as $TD_p(u) := \sum_v p(v)d_G(u, v)$). Note that the total distance for unweighted graphs is nothing but the inverse of closeness centrality. Our approach can also be applied to that case (Theorem 17).

Finally, as a byproduct of our techniques, we obtain a new *distance labeling scheme* for bounded clique-width graph classes which outperforms the state of the art [20][1]. See our Theorem 4 for details. In doing so, we get an $\tilde{\mathcal{O}}(kn^2)$-time algorithm in order to solve All-Pairs Shortest-Paths within n-vertex vertex-weighted graphs of clique-width at most k (Corollary 7). This improves on the previously best-known $\mathcal{O}((kn)^2)$-time algorithm, and it almost completely solves an open problem from Kratsch and Nelles [47] who asked whether there exists an $\mathcal{O}(kn^2)$-time algorithm for this problem.

Overview of our techniques. Roughly, the standard approach for bounded clique-width graphs is to process a k-expression sequentially. It is possible to transform a k-expression into a so called *partition tree*, a purely combinatorial object that has been used in [18] in order to derive a new characterization of the clique-width. – We formally define clique-width and partition trees in Sec. 2. – Doing so, it becomes easier and more transparent to apply standard algorithmic approaches, for trees, to the k-expressions. In particular, it is known

[1] In all fairness, the labeling scheme of Courcelle and Vanicat can be applied to many more problems than just the computation of the distances in the graph.

that every bounded clique-width graph has a balanced edge-cut of bounded neighbourhood diversity (*i.e.*, whose edges can be partitioned in a bounded number of complete bipartite graphs) [3, 26]. As a side contribution of this work, we show how to compute such balanced cuts in parameterized linear time from a given partition tree. – Note that the original runtime from [3] is unknown to us as we were unable to find this reference. – This procedure of recursively finding such an edge-cut produces a special type of centroid decomposition of a partition tree, with algorithmic applications to several distance problems on bounded clique-width graphs. While such a divide-and-conquer approach can hardly be considered as "new", its usefulness in the fine-grained complexity study of polynomial-time solvable problems on bounded clique-width graphs has remained to be demonstrated until our work. We expect several other results to be found with this approach, in a similar way to what has been done for bounded treewidth graphs in [44].

The distance-labeling scheme of Theorem 4 follows almost directly from our centroid decomposition of a partition tree, that is why we chose to present it first in the paper. In order to compute the centrality indices, we combine this centroid decomposition with two other tools. One is the range query framework of Cabello and Knauer [13] that we use to compute some distance information (depending on the centrality index) between the vertices that are on different sides of an edge-cut of small neighbourhood diversity. To our best knowledge, our work is the first (but admittedly, simple) application of this framework to edge-cuts. We also augment this framework with a new type of orthogonal range query, with applications to the fast computation of all p-eccentricities and total p-distances, see Sec. 5. Our second tool is inspired from prior works on bounded treewidth graphs [1, 11] and Cunningham's split decomposition [21]. Specifically, we design some *edge*-weighted gadgets in order to preserve the distances of the original graph in the two subgraphs resulting from the removal of an edge-cut of bounded neighbourhood diversity. Adding weighted edges is problematic because the diameter problem cannot even be solved in truly subquadratic time within edge-weighted graphs of bounded clique-width. To address this issue, we restrict our addition of weighted edges to ensure that when we further partition the graph via more edge-cuts, the weighted edges are not included in these edge-cuts. To do this, we partition the vertices of our gadgets into at most $\mathcal{O}(\log n)$ clusters of only $\mathcal{O}(k^2)$ vertices each, so that weighted edges are only added between pairs of vertices in the same cluster. Then, we ensure that no cluster is ever separated by an edge-cut computed from the partition tree. Doing so, we are still able to ensure that we can find *unweighted* edge-cuts that satisfy the requirement of being both balanced and of small neighborhood diversity. We stress that to prove correctness of our construction, we had to carefully analyze the structure of a partition tree, which is arguably the most technical part of our analysis. While it is tempting to make our gadgets vertex-weighted (*e.g.*, by properly subdividing the weighted edges), we did not find a satisfying way to do that without increasing the neighbourhood diversity of some of the cuts.

Notations. Throughout the remainder of the paper, we shall write $G = (V, E)$ for an unweighted graph, and $G = (V, E, w)$ for a vertex-weighted graph, where $w : V \to \mathbb{N}$. The neighbour set of a vertex $v \in V$, resp. of a subset $S \subseteq V$, is defined as $N_G(v) = \{u \in V \mid uv \in E\}$, resp. as $N_G(S) = \bigcup_{v \in S} N_G(v) \setminus S$. We may also define the distance between a vertex $v \in V$ and a subset $S \subseteq V$ as $d_G(v, S) = d_G(S, v) = \min_{u \in S} d_G(u, v)$, and the distance between two subsets S, S' as $d_G(S, S') = \min_{u \in S, v \in S'} d_G(u, v)$. Note that if $S = \emptyset$ then, $d_G(v, S) = d_G(S, S') = +\infty$ for any v and S'. We shall introduce additional terminology wherever needed in the paper.

2 Clique-width and partition trees

First, we recall two equivalent definitions of clique-width [18]. The following definitions can be extended to weighted graphs simply by ignoring all the weights.

Clique-width expressions. A k-labeled graph is a triple $G = (V, E, \ell)$ where $\ell : V \to \{1, 2, \ldots, k\}$ is called a labeling function. A clique-width k-expression (for short, a k-expression) is an algebraic expression where the four allowed operations are: $i(v)$: we add a new isolated vertex with label $\ell(v) = i$; $G_1 \oplus G_2$: we make the disjoint union of two k-labeled graphs; $\eta(i, j)$: we add a join (complete bipartite subgraph) between all vertices with label i and all vertices with label j; $\rho(i, j)$: for all vertices v s.t. $\ell(v) = i$, we set $\ell(v) = j$. The generated graph is the one obtained from the k-expression by deleting all the labels. We say that a graph $G = (V, E)$ has clique-width at most k if it is the graph generated by some k-expression. For instance, $1(a)2(b)\eta(1, 2)\rho(1, 3)1(c)\eta(1, 2)\rho(2, 3)2(d)\eta(1, 2)$ is a 3-expression generating the four-node path P_4 with nodes a, b, c, d. In particular, the clique-width of P_4 is at most three. This is in fact an equality, as the graphs of clique-width at most two are exactly the cographs [40]. We denote by $cw(G)$ the clique-width of the graph G. The size of a k-expression is its number of operations. If the generated graph has order n and m edges, and there is no unnecessary operation $\rho(i, j)$ nor $\eta(i, j)$ – which we will assume to be the case throughout the remainder of this paper –, then the k-expression has size in $\mathcal{O}(n + m)$ (*e.g.*, see [35], where Fürer proved this result in a more general setting).

Partition tree. It is useful to represent a k-expression as a parse tree. By iteratively contracting the edges incident to non-branching nodes of a parse tree, we get a so-called partition tree, whose nodes are mapped to the collection of subsets of vertices with equal label in their rooted subtree. Formally, given a graph $G = (V, E)$, a partition tree is a pair (T, f) where T is a rooted tree whose inner nodes have at least two children, and f is a function mapping every node of T to a partial partition of V, such that:

- for every node $a \in V(T)$, $f(a)$ is a partition of some vertex-subset $A \subseteq V$;
- for every vertex $v \in V$, there is a leaf node $a_v \in V(T)$ s.t. $f(a_v) = \{\{v\}\}$;
- for every inner node $a \in V(T)$, let $b_1, b_2, \ldots, b_d$ be its children. If $f(a)$ is a partition of A, and in the same way for every $1 \leq i \leq d$, $f(b_i)$ is a partition of B_i, then the vertex-subsets $B_1, B_2, \ldots, B_d$ are pairwise disjoint and $A = \bigcup_{i=1}^{d} B_i$. Furthermore, for every $1 \leq i \leq d$, for every subset $X_i \in f(b_i)$, there is $X \in f(a)$ s.t. $X_i \subseteq X$ (we say that $\bigcup_{i=1}^{d} f(b_i)$ refines $f(a)$). Finally, for every $1 \leq i < j \leq d$, for every adjacent vertices $v_i \in B_i$ and $v_j \in B_j$, if $v_i \in X$ and $v_j \in Y$, for some $X, Y \in f(a)$, then we have $X \neq Y$ and $X \times Y \subseteq E$ (we say that the partition is compatible with the edge-incidence relation in the graph G).

The *width* of a partition tree is equal to $\max_{a \in V(T)} |f(a)|$. A graph has clique-width at most k if and only if it admits a partition tree of width at most k [18].

Note that if we naively store a partition tree (T, f), then storing explicitly all the labels $f(a)$, for $a \in V(T)$, would require $\mathcal{O}(n^2)$ space. Instead, for every $a \in V(T)$, for every $X \in f(a)$, we may create a new vertex (a, X); then if b_i is a child of a, for every $X_i \in f(b_i)$ s.t. $X_i \subseteq X$, we add an arc between (a, X) and (b_i, X_i). This is called in [18] the representation graph of (T, f) and it only requires $\mathcal{O}(kn)$ space if the width is at most k.

▶ **Lemma 1** ([18]). *There is an algorithm that transforms a k-expression of size L into the representation graph of a width-k partition tree in $\mathcal{O}(kL)$ time.*

In particular, given a k-expression for an n-vertex m-edge graph G, we can construct the representation graph of a width-k partition tree in $\mathcal{O}(k(n + m))$ time.

Relation with k-modules. For a graph $G = (V, E)$, a subset $M \subseteq V$ is a module if we have $N_G(u) \setminus M = N_G(v) \setminus M$ for every vertices $u, v \in M$. A k-module is some $M \subseteq V$ that can be partitioned into k subsets, denoted $M_1, M_2, \ldots, M_k$, in such a way that for every $1 \leq i \leq k$, M_i is a module in the subgraph $G[(V \setminus M) \cup M_i]$. Some relations between clique-width and k-modules were explored in [51]. We make the following useful observation, whose proof is inspired by [51, Theorem 7].

▶ **Lemma 2.** *The following two properties hold for every partition tree (T, f) of a graph $G = (V, E)$:*

1. *For every node $a \in V(T)$, let $A = \bigcup f(a)$ be the vertex-subset of which $f(a)$ is a partition. Then, A is a $|f(a)|$-module of G, with a corresponding partition of A being $f(a)$.*
2. *Let $a_1, a_2, \ldots, a_p$ be some children nodes of some $a' \in V(T)$ and, for each $1 \leq i \leq p$, let $A_i = \bigcup f(a_i)$ be the vertex-subset of which $f(a_i)$ is a partition. Then, $A = \bigcup_{i=1}^{p} A_i$ is a $|f(a')|$-module of G, with a corresponding partition of A being $\{X' \cap A \mid X' \in f(a')\}$.*

Finally, recall that a cut of $G = (V, E)$ is a bipartition $(A, V \setminus A)$ of its vertex-set. The *neighbourhood diversity* of a cut is the least k s.t. A is a k-module of G. By Lemma 2, each node of a width-k partition tree defines a cut of neighbourhood diversity at most k.

3 Distance-labeling scheme

We describe our distance oracle for bounded clique-width graph classes. For technical reasons, we need to make it work also for unconnected graphs. While it is likely that we could process each connected component separately, we did not explore this possibility since it was leading to more complicated updates of the partition trees (see the proof of Theorem 4 below). Given a possibly unconnected graph G, the *distance* $d_G(u, v)$ between $u, v \in V$ is equal to: $+\infty$ if u and v are on different connected components of G, and to the smallest weight of a uv-path in G otherwise. A *distance-labeling scheme* consists in some encoding function $C_G : V \to \{0,1\}^*$ and some decoding function $D_G : \{0,1\}^* \times \{0,1\}^* \to \mathbb{N} \cup \{+\infty\}$ s.t. $d_G(u, v) = D_G(C_G(u), C_G(v))$ for every vertices u and v. We are interested in minimizing the total pre-processing time in order to compute the labels $C_G(v)$, for all vertices v, and the query time in order to compute the distance given two labels. It is often the case that D_G runs in time polynomial in the size of the labels. Then, the objective is to minimize $\max_{v \in V} |C_G(v)|$. The following result is due to Courcelle and Vanicat:

▶ **Theorem 3** ([20])**.** *The family of n-vertex bounded clique-width unweighted graphs enjoys an exact distance labeling scheme using labels of length $\mathcal{O}(\log^2 n)$ bits. Moreover, the distance between two vertices can be computed in $\mathcal{O}(\log^2 n)$ time.*

The hidden dependency in the clique-width is a stack of exponentials [37]. We improve the latter while keeping optimal bit size and improved query time, namely:

▶ **Theorem 4.** *For a vertex-weighted graph, let W denote the maximum weight. The family of n-vertex m-edge graphs of clique-width at most k enjoys an exact distance labeling scheme using labels of length $\mathcal{O}(k \log n \log(nW))$ bits (resp., $\mathcal{O}(k \log^2 n)$ bits if the graph is unweighted). Moreover, all the labels can be pre-computed in $\mathcal{O}(k(n+m) \log^2 n)$ time if a k-expression is given (resp., in $\mathcal{O}(k(n+m) \log n)$ time if the graph is unweighted), and the distance between two vertices can be computed in $\mathcal{O}(k \log n)$ time.*

For the related problem of adjacency queries, we refer to [45] for a data structure in $\mathcal{O}(kn)$ space for the n-vertex graphs of clique-width at most k.

Recall that $d_G(v,S) = d_G(S,v) = \min_{u \in S} d_G(u,v)$. In particular, $d_G(v,S) = +\infty$ if S is empty. We will need the following result:

▶ **Lemma 5.** *Let $G = (V,E,w)$ be a graph (possibly not connected) and let $(A, V \setminus A)$ be a cut of neighbourhood diversity k. Furthermore, let $A_1, A_2, \ldots, A_k$ be a partition of A s.t. for every $1 \leq i \leq k$, A_i is a module of $G \setminus (A \setminus A_i)$. For $1 \leq i \leq k$, let $B_i = N_G(A_i) \setminus A$. The following hold for every $u, v \in V$:*

- *if $u \in A$, $v \notin A$ then $d_G(u,v) = \min\{d_G(u,A_i) + d_G(B_i,v) \mid 1 \leq i \leq k\}$;*
- *if $u,v \in A$ then $d_G(u,v) = \min\{d_{G[A]}(u,v)\} \cup \{d_G(u,A_i) + d_G(B_i,v) \mid 1 \leq i \leq k\}$;*
- *if $u,v \notin A$ then $d_G(u,v) = \min\{d_{G[V \setminus A]}(u,v)\} \cup \{d_G(u,A_i) + d_G(B_i,v) \mid 1 \leq i \leq k\}$.*

Our scheme for bounded clique-width graphs mimics one very well-known for trees which is based on the centroid decomposition [36]. Specifically, let $w : V(T) \to \mathbb{N}$ assign non-negative weights to the nodes of some tree T. A w-centroid is a node c s.t. every subtree of $T \setminus \{c\}$ has weight at most $w(T)/2$. Such node always exists and a centroid can be computed in linear time by using a standard dynamic programming approach [39] (simply orient each edge toward the heaviest subtree, then find a sink). We also need the following easy lemma:

▶ **Lemma 6.** *If c is a w-centroid of a tree T, then the components of $T \setminus \{c\}$ can be partitioned in linear time in two forest F_1, F_2 s.t. $\max\{w(F_1), w(F_2)\} \leq 2w(T)/3$.*

We are now ready to prove the main result of this section:

Proof of Theorem 4. We fix some width-k partition tree (T,f), that takes $\mathcal{O}(k(n+m))$ time to compute by using Lemma 1. Let $w : V(T) \to \{0,1\}$ be s.t. $w(a) = 1$ if and only if a is a leaf. Observe that $w(T) = n$ since there is a one-to-one mapping between the vertices in V and the leaves of T. In order to construct the labels $C_G(v)$, for all $v \in V$ (encoding function), we next define a recursive procedure onto the weighted partition tree.

In what follows, let us assume $n > 1$ (otherwise, there is nothing to be done). We compute in $\mathcal{O}(|V(T)|)$ time, and so in $\mathcal{O}(n)$ time, a w-centroid c. Note that if $n = 2$, then T is composed of a root and of two leaves; then, a good choice for the w-centroid c is to take the root. In particular, we may assume c to be an internal node. Otherwise, $n \geq 3$, and so, since $w(T) = n$, we *must* have that c is an internal node. Then, let $a_1, a_2, \ldots, a_d$ be the children of c. We denote C (resp. A_i) the subset of vertices of which $f(c)$ (resp., $f(a_i)$) is a partition. Furthermore, let T_c (resp., let T_{a_i}) be the subtree rooted at c (resp., at a_i). By Lemma 6 we can bipartition the trees $T \setminus T_c$, $T_{a_1}, T_{a_2}, \ldots, T_{a_d}$ into two forests F_1, F_2 of respective total weights $\leq 2n/3$. In particular, since c is internal, and so $w(c) = 0$, both forests are non-empty. Up to re-ordering the children nodes of c, we may assume one of those forests, say F_1, to be equal to $\bigcup_{j=1}^{p} T_{a_j}$, for some $p \leq d$. For short, we name $A := \bigcup_{j=1}^{p} A_j$. Doing so, we define the cut $(A, V \setminus A)$, whose two sides can be determined in $\mathcal{O}(n)$ time by traversing the disjoint subtrees $T_{a_1}, T_{a_2}, \ldots, T_{a_p}$.

By Lemma 2, A is a k-module of G, with a corresponding partition being $\Phi(A) = \{X \cap A \mid X \in f(c)\}$ (or $f(a_1)$ if $p = 1$). Note that such a partition can be readily derived in $\mathcal{O}(n)$ time from either $f(c)$ or $f(a_1)$. In turn, being given the representation graph of (T,f), we can compute $f(c)$ and $f(a_1)$ in $\mathcal{O}(kn)$ time by traversing the subtrees rooted at nodes c and a_1. Let $X_1, X_2, \ldots, X_k$ be a partition of A s.t., for every $1 \leq i \leq k$, X_i is a module of $G \setminus (A \setminus X_i)$. Furthermore, for every $1 \leq i \leq k$, let $Y_i := N_G(X_i) \setminus A$ (neighbour sets in $V \setminus A$). Since the subsets X_i are pairwise disjoint we can compute $Y_1, Y_2, \ldots, Y_k$ in total $\mathcal{O}(m)$ time. Finally, for every $1 \leq i \leq k$, for every $v \in V$, we compute $d_G(v, X_i)$ and $d_G(v, Y_i)$. It takes $\mathcal{O}((m+n)\log n)$ time per subset, using a modified Dijkstra's algorithm, and so total time in $\mathcal{O}(k(m+n)\log n)$ (resp., if the graph is unweighted, then it takes $\mathcal{O}(m+n)$ time per subset,

using a modified BFS, and so total time in $\mathcal{O}(k(m+n))$). We end up applying recursively the same procedure as above on the disjoint (possibly unconnected) subgraphs $G[A]$ and $G[V \setminus A]$. For that, we need to build a partition tree for each subgraph.

- For $G[A]$, we take $T_A = T_{a_1}$ if $p = 1$, otherwise we take $T_A = T_c \setminus (\bigcup_{j>p} T_j)$. Then, for every $b \in V(T_A)$, we set $f_A(b) = \{X \cap A \mid X \in f(b)\}$. Observe that if $b \neq c$ then $f_A(b) = f(b)$. Hence, the representation graph of (T_A, f_A) can be computed from the representation graph of (T, f) in $\mathcal{O}(kn)$ time.
- For $G[V \setminus A]$, a natural choice would be to take the subtree $T_{V\setminus A} = T \setminus (\bigcup_{j=1}^{p} T_{a_j})$. Then, for every $b \in V(T_{V\setminus A})$, we set $f_{V\setminus A}(b) = \{X \setminus A \mid X \in f(b)\}$. Again, we observe that the representation graph of $(T_{V\setminus A}, f_{V\setminus A})$ can be computed from the representation graph of (T, f) in $\mathcal{O}(kn)$ time. However, doing so, we may not respect all properties of a partition tree. Specifically, if $d = p$ then c has become a leaf-node and it must be removed. But then, its father node c' may have only one child b left. If that is the case, then either c' is the root of T and then we choose $T_{V\setminus A} = T_b$, or we choose the father node of c' as the new father node of b, removing on our way the node c'. Note that we do *not* modify $f_{V\setminus A}(b)$ during this procedure. Finally, if $d = p + 1$ then c only has one child a_d left. We proceed similarly as in the previous case. That is, either c was the root of T and then we set $T_{V\setminus A} = T_{a_d}$, or we choose the father node of c as the new father node of a_d, removing on our way the node c. Note that doing so, we do *not* modify the partition $f_{V\setminus A}(a_d)$.

The above procedure recursively defines a so called w-centroid decomposition $T^{(w)}$. The latter is a binary rooted tree, whose root is labeled by the cut $(A, V \setminus A)$. Its left and right subtrees are w-centroid decompositions of $G[A]$ and $G[V \setminus A]$ respectively. Note that by construction, the depth of $T^{(w)}$ is in $\mathcal{O}(\log n)$. Furthermore, there is a one-to-one mapping between the leaves of $T^{(w)}$ and the vertices of G. For every vertex $v \in V$, its label $C_G(v)$ contains each cut on its path until the root of $T^{(w)}$, and the $2k$ distances computed for each cut. – Infinite distances may be encoded as some special character. – Here, we stress that all these distances are computed in some induced subgraphs of G, and not in G itself (unless it is for the first cut, at the root). Since the depth of $T^{(w)}$ is in $\mathcal{O}(\log n)$, each $C_G(v)$ stores $\mathcal{O}(k \log n)$ distances, and so it has a bit size in $\mathcal{O}(k \log n \log (Wn))$ (resp, in $\mathcal{O}(k \log^2 n)$ if the graph is unweighted). Furthermore, as $G[A]$ and $G[V \setminus A]$ are disjoint, every recursive stage of the procedure takes $\mathcal{O}(k(n+m) \log n)$ time (resp., $\mathcal{O}(k(n+m))$ time). Hence, the total pre-processing time in order to compute $C_G(v)$, for all $v \in V$, is in $\mathcal{O}(k(n+m) \log^2 n)$ (resp., in $\mathcal{O}(k(n+m) \log n)$ if G is unweighted).

We are left describing D_G (decoding). Let $u, v \in V$ be arbitrary. Their least common ancestor in $T^{(w)}$ corresponds to some cut $(A^j, A^{j-1} \setminus A^j)$ s.t. $u \in A^j$, $v \in A^{j-1} \setminus A^j$. Consider *all* the cuts on the path between their least common ancestor and the root of $T^{(w)}$. We call the latter $(A^0, V \setminus A^0), (A^1, A^0 \setminus A^1), \ldots, (A^j, A^{j-1} \setminus A_j)$. Since up to reverting their two sides, all these cuts have neighbourhood diversity at most k, then we may apply Lemma 5 $j + 1$ times in order to compute $d_G(u, v)$ (*i.e.*, in $G, G[A^0], G[A^1], \ldots, G[A^{j-1}]$). Note that $j = \mathcal{O}(\log n)$. Finally, since for each cut considered, the $2k$ distances that are required in order to apply this lemma are stored in $C_G(u)$ and $C_G(v)$, it takes $\mathcal{O}(k)$ time per cut, and so, the final query time is in $\mathcal{O}(k \log n)$. ◀

Recall that All-Pairs Shortest-Paths in an n-vertex graph of clique-width at most k can be solved in $\mathcal{O}((kn)^2)$ time [47]. As a by-product of our Theorem 4, we observe below that we can improve the dependency on k, but at the price of a poly-logarithmic overhead in the running time.

▶ **Corollary 7.** *For every n-vertex vertex-weighted graph $G = (V, E, w)$, if $cw(G) \leq k$ and a k-expression is given, then we can solve All-Pairs Shortest-Paths for G in $\mathcal{O}(k(n \log n)^2)$ time (resp., in $\mathcal{O}(kn^2 \log n)$ time if G is unweighted).*

Proof. We start applying Theorem 4 in order to compute a distance-labeling scheme with $\mathcal{O}(k \log n)$ query time. Then, we consider all pairs $u, v \in V$ (there are $\mathcal{O}(n^2)$ such pairs) and we compute $d_G(u, v)$ in $\mathcal{O}(k \log n)$ time. ◀

4 Centrality indices and beyond

We refine our strategy for Theorem 4 in order to prove the main result of this paper:

▶ **Theorem 8.** *For every connected n-vertex m-edge graph $G = (V, E, w)$, if $cw(G) \leq k$ and a k-expression is given, then we can compute in $\mathcal{O}(2^{\mathcal{O}(k)}(n+m)^{1+\epsilon})$ time, for any $\epsilon > 0$: all the eccentricities, and all the closeness centralities.*

▶ **Corollary 9.** *For every connected n-vertex m-edge graph $G = (V, E, w)$, if $cw(G) \leq k$ and a k-expression is given, then we can compute the diameter, radius, center, Wiener index and median set of G in $\mathcal{O}(2^{\mathcal{O}(k)}(n+m)^{1+\epsilon})$ time, for any $\epsilon > 0$.*

Recall that Coudert et al. proved that assuming SETH, for any $\epsilon > 0$, there is no $\mathcal{O}(2^{o(k)}(n+m)^{2-\epsilon})$-time algorithm for computing the diameter within cubic graphs of clique-width at most k [16]. Therefore, our results for the diameter (and so, for the eccentricities) are optimal under SETH. Our results for the Wiener index (and so, for the closeness centrality) are also optimal under SETH. Indeed, since the pathwidth of a graph is an upper bound for its clique-width [30], then it follows from [1] that it is already "SETH-hard", in the unweighted case, to decide in $\mathcal{O}(2^{o(k)}(n+m)^{2-\epsilon})$ time whether the diameter is either two or three. It is well-known that $diam(G) \leq 2$ if and only if for every $v \in V$ of degree $\delta_G(v)$, $TD(v) = 2(n-1) - \delta_G(v)$ [11]. In particular, $diam(G) \leq 2$ if and only if $W(G) = 2n(n-1) - 2m$.

4.1 Minimal partition of k-modules

First, it is not hard to show that every k-module has a partition in a least number of subsets. In what follows, we will often use a few simple properties of this minimal partitioning.

▶ **Lemma 10.** *Every vertex-subset A in a graph $G = (V, E, w)$ admits a unique partition $A_1, A_2, \ldots, A_k$ with the following two properties:*

1. *For every $1 \leq i \leq k$, for every $u_i, v_i \in A_i$, we have $N_G(u_i) \setminus A = N_G(v_i) \setminus A$. In particular, A is a k-module of G.*
2. *For every $k' < k$, A is* not *a k'-module of G.*

We call it the minimal partition of A, and it can be computed in linear time.

Additional notations. From this point on we need to also allow *edge-weights*, due to some technicalities in our final proof of Theorem 8. Such a graph is denoted by $G = (V, E, w, \alpha)$, where $\alpha : E \to \mathbb{N}$. Then, the weight of a path is the sum of the weights of all its vertices and edges. The special case of vertex-weighted graphs is retrieved by setting $\alpha(e) = 0$ for every $e \in E$. Finally, we call a cut $(A, V \setminus A)$ *unweighted* if all edges between A and $V \setminus A$ have a zero weight. The neighbourhood diversity of a cut is the same in G as in the underlying unweighted graph obtained from G by removing all the weights.

4.2 Orthogonal range queries

We then need to recall some basics about the framework introduced in [13] by Cabello and Knauer. Let $P \subseteq \mathbb{R}^k$ be a static set of k-dimensional points. We assume each point $\overrightarrow{p} \in P$ to be assigned a value $g(\overrightarrow{p})$. A box is the Cartesian product of k intervals. Note that we allow each interval to be unbounded and/or open or partially open. Roughly, given a box $\mathcal{R}$, a range query on P asks for either reporting or counting all points in $P \cap \mathcal{R}$, or for some specific point(s) in this intersection maximizing a given objective function. Here, we consider the following types of range queries:

- (*Maximum range query*) Given some box $\mathcal{R}$, find some $\overrightarrow{p} \in P \cap \mathcal{R}$ maximizing $g(\overrightarrow{p})$;
- (*Sum range query*) Given some box $\mathcal{R}$, compute $\sum_{\overrightarrow{p} \in P \cap \mathcal{R}} g(\overrightarrow{p})$.
- (*Count range query*) Given some box $\mathcal{R}$, compute $|P \cap \mathcal{R}|$.

▶ **Lemma 11** ([11]). *For every k-dimensional point set P of size n, for any $\epsilon > 0$, we can construct in $\mathcal{O}(2^{\mathcal{O}(k)} n^{1+\epsilon})$ time, a data structure, sometimes called a k-dimensional range tree, that allows to answer any maximum range query, sum range query or count range query in $\mathcal{O}(2^{\mathcal{O}(k)} n^{\epsilon})$ time.*

In the following Lemma 12 we give a new simple application of Lemma 11 to distance problems in graphs, namely:

▶ **Lemma 12.** *Let $G = (V, E, w, \alpha)$ be a connected n-vertex m-edge graph, let $(A, V \setminus A)$ be an unweighted cut of neighbourhood diversity at most k, and let $A' \subseteq A$, $B' \subseteq V \setminus A$. For any $\epsilon > 0$, after a pre-processing in $\mathcal{O}(km + 2^{\mathcal{O}(k)} n^{1+\epsilon})$ time, for every vertex $u \in A'$ we can compute the values $\max_{v \in B'} d_G(u, v)$ and $\sum_{v \in B'} d_G(u, v)$ in $\tilde{\mathcal{O}}(2^{\mathcal{O}(k)} n^{\epsilon})$ time; in the same way, for every vertex $v \in B'$ we can compute the values $\max_{u \in A'} d_G(v, u)$ and $\sum_{u \in A'} d_G(v, u)$ in $\mathcal{O}(2^{\mathcal{O}(k)} n^{\epsilon})$ time.*

4.3 Distance-preservers with weighted edges

Our next objective consists in adding some weighted subsets to the two sides of a cut in order to preserve the distances from the original graph. Recall that for every two subsets X and Y, $d_G(X, Y) = \min_{x \in X, y \in Y} d_G(x, y)$. Our construction below is inspired by Cunningham's split decomposition [21].

▶ **Definition 13.** *Given $G = (V, E, w, \alpha)$ connected, let $(A, V \setminus A)$ be an unweighted cut of neighbourhood diversity at most k. Let $A_1, A_2, \ldots, A_k$ be the minimal partition of A. W.l.o.g., either all the B_i's are nonempty, or B_k is the unique empty set amongst the B_i's. We set $k' = k$ if $B_k \neq \emptyset$, and $k' = k - 1$ otherwise.*

- *For every $1 \leq i \leq k'$, let $b_{ii} \in B_i$ be of minimum weight. For every $1 \leq i < j \leq k'$, let also $b_{ij} \in B_i$, $b_{ji} \in B_j$ be the ends of a shortest $B_i B_j$-path (possibly, $b_{ij} = b_{ji}$). The graph H_A is obtained from $G[A \cup \{b_{ij} \mid 1 \leq i, j \leq k'\}]$ by adding, for every $1 \leq i < j \leq k'$ s.t. $b_{ij} \neq b_{ji}$, an edge $b_{ij} b_{ji}$ of weight $d_G(B_i, B_j) - w(b_{ij}) - w(b_{ji})$.*
- *For every $1 \leq i \leq k'$, let $a_{ii} \in A_i$ be of minimum weight. For every $1 \leq i < j \leq k'$, let also $a_{ij} \in A_i$, $a_{ji} \in A_j$ be the ends of a shortest $A_i A_j$-path. The graph H_B is obtained from $G[(V \setminus A) \cup \{a_{ij} \mid 1 \leq i, j \leq k'\}]$ by adding, for every $1 \leq i < j \leq k'$, an edge $a_{ij} a_{ji}$ of weight $d_G(A_i, A_j) - w(a_{ij}) - w(a_{ji})$.*

Below, we observe that it is rather straightforward to compute these two above subgraphs H_A and H_B in parameterized almost linear time:

▶ **Lemma 14.** *Given $G = (V, E, w, \alpha)$ connected, let $(A, V \setminus A)$ be an unweighted cut of neighbourhood diversity at most k. The gadget subgraphs H_A and H_B (see Definition 13) can be constructed in $\tilde{\mathcal{O}}(k^2 n + km)$ time.*

The following two properties are crucial in our proofs of Theorem 8.

▶ **Lemma 15.** *Given $G = (V, E, w, \alpha)$ connected, let $(A, V \setminus A)$ be an unweighted cut of neighbourhood diversity at most k. Let H_A, H_B be as in Definition 13. Then, for every $u, v \in A$ we have $d_G(u, v) = d_{H_A}(u, v)$. Similarly, for every $u, v \notin A$ we have $d_G(u, v) = d_{H_B}(u, v)$.*

Our approach only works for unweighted cuts. In particular, if we want to apply the procedure of Definition 13 recursively, for some cuts in the gadget subgraphs H_A and H_B, then we must have both ends of each weighted edge on a same side of the cut. The next lemma shows that restricting ourselves to such cuts does not cause an explosion of their neighbourhood diversity.

▶ **Lemma 16.** *Given $G = (V, E, w, \alpha)$ connected, let $(A, V \setminus A)$ be an unweighted cut of neighbourhood diversity at most k. Let H_A, H_B be as in Definition 13.*

1. *For any $A' \subseteq A$, if A' is a k-module of G then it is a k-module of H_A.*
2. *For any $B' \subseteq V \setminus A$, if B' is a k-module of G then it is a k-module of H_B; if $A \cup B'$ is a k-module of G then $B' \cup \{a_{ij} \mid 1 \leq i, j \leq k'\}$ is a k-module of H_B.*

4.4 Proofs of the main results

Sketch Proof of Theorem 8. We revisit the scheme of Theorem 4. That is, we fix some width-k partition tree (T, f), that takes $\mathcal{O}(k(n+m))$ time by using Lemma 1. We pre-process the tree T in order to compute in $\mathcal{O}(1)$ time, for any two nodes $a, a' \in V(T)$, their least common ancestor; it can be done in $\mathcal{O}(n)$ time [43]. Furthermore, let $w : V(T) \to \{0, 1\}$ be s.t. $w(a) = 1$ if and only if a is a leaf. In what follows, we mimic the recursive construction of a w-centroid decomposition of T.

The algorithm. We consider a more general problem for which we are given as input some tuple $\langle r, H, U, T^U, f^U, \mathcal{L}\rangle$. Let us detail each of the components of this input. Here, H is a graph with non-negative real vertex-weights and non-negative integer edge-weights (initially, $H = G$). The value r represents the recursion level of the algorithm (initially, $r = 0$). The vertex-subset U is such that $U \subseteq V \cap V(H)$ (initially, $U = V$). We further impose to have $H[U] = G[U]$, and that for every $u, v \in U$ we have $d_G(u, v) = d_H(u, v)$. In particular, all the edges of $H[U]$ are unweighted. The rooted tree (T^U, f^U) is a width-k-partition tree of $G[U]$ (initially, $T^U = T$ and $f^U = f$). We further assume that T^U was constructed from a rooted subtree of T by repeatedly contracting internal nodes with only one child. In particular, all the ancestor-descendant relations in T^U are also ancestor-descendant relations in T. Furthermore, for every node $b \in V(T^U)$ we impose $f^U(b) = \{X \cap U \mid X \in f(b)\}$. Note that in lieu of (T^U, f^U), we are given the representation graph of this partition tree (as defined in Sec. 2). Finally, $H \setminus U$ is partitioned in $r' \leq r$ subgraphs of order $\mathcal{O}(k^2)$, that we shall name "clusters" in what follows (there may exist edges between different clusters, but they must be unweighted). To each cluster W_i, we associate some node c_i of the original tree T. Roughly, c_i corresponds to some balanced cut, computed at an earlier recursive stage, and the cluster W_i resulted from the procedure of Definition 13 applied to this cut. So, in particular, we impose that any edge between two vertices that are on different clusters (resp., between a vertex in a cluster and a vertex of U) must be unweighted. All the pairs (W_i, c_i) are stored in the list $\mathcal{L}$ (initially, $\mathcal{L}$ is the empty list).

The output of the algorithm is, for each $u \in U$, $\max_{v \in U} d_H(u, v)$ and $\sum_{v \in U} d_H(u, v)$.

We may assume that $|U| \geq \lambda k^2 \log n$, for some sufficiently large constant λ. Indeed, if it not the case then we may compute by brute-force all the desired values (base case of the recursion). Then, we compute a w-centroid c in T^U. Since $w(T^U) = |U| > 3$, this node c cannot be a leaf. Let $a_1, a_2, \ldots, a_d$ be the children of c. As before, we denote by C (resp. A_i) the subset of vertices of which $f^U(c)$ (resp., $f^U(a_i)$) is a partition, and by T^U_c (resp., $T^U_{a_i}$) the subtree rooted at c (resp., at a_i). Here, we stress that $C \subseteq U$ (resp., $A_i \subseteq U$). By using Lemma 6, we may partition $T^U \setminus \{c\}$ in two non-empty forests of respective weights $\leq 2|U|/3$. Furthermore, we may assume one of our two forests to contain exactly $T^U_{a_1}, T^U_{a_2}, \ldots, T^U_{a_p}$ for some $p \leq d$. Then, let $A = \bigcup_{j=1}^{p} A_j$. We compute the following cut of H:

- The subsets A and $U \setminus A$ are on separate sides of the cut.
- For every $(W_j, c_j) \in \mathcal{L}$, there are two cases. If there exists some index $1 \leq i \leq p$ s.t. the least common ancestor of c_j and a_i in T is a *strict* descendant of c (a child of c in T, or a descendant of one of these children), then we put W_j on the same side of the cut as A. Otherwise, we put W_j on the same side of the cut as $U \setminus A$.

Note that, for each $(W_j, c_j) \in \mathcal{L}$, we can decide in which case we are as follows. For every $1 \leq i \leq p$, we compute the least common ancestor s_i of c_j and a_i in T. Then, for every $1 \leq i \leq p$, we compute the least common ancestor of s_i and c in T.

Let $(A', V(H) \setminus A')$ be the resulting cut, where $A \subseteq A'$. By construction, it is unweighted. We prove that A' is a k-module of H. Then, we apply Lemma 12 in order to compute, for every $u \in A$, the values $\max_{v \in U \setminus A} d_H(u, v)$ and $\sum_{v \in U \setminus A} d_H(u, v)$ (resp., for every $v \in U \setminus A$, the values $\max_{u \in A} d_H(v, u)$ and $\sum_{u \in A} d_H(v, u)$). We are left computing for every $u \in A$, the values $\max_{u' \in A} d_H(u, u')$ and $\sum_{u' \in A} d_H(u, u')$ (resp., for every $v \in U \setminus A$, the values $\max_{v' \in U \setminus A} d_H(v, v')$ and $\sum_{v' \in U \setminus A} d_H(v, v')$). For that, we construct the gadget subgraphs H_A and H_B, as in Definition 13 (*i.e.*, w.r.t. the above cut $(A', V(H) \setminus A')$). Let $\mathcal{L}_A$ contain every $(W_j, c_j) \in \mathcal{L}$ s.t. $W_j \subseteq A'$; we also add in $\mathcal{L}_A$ a new cluster $(V(H_A) \setminus A', c)$. In the same way, let $\mathcal{L}_B$ contain every $(W_j, c_j) \in \mathcal{L}$ s.t. $W_j \subseteq V(H) \setminus A'$; we also add in $\mathcal{L}_B$ a new cluster $(V(H_B) \setminus B', c)$, where $B' = V(H) \setminus A'$. We end up calling our algorithm recursively for the inputs $\langle r+1, H_A, A, T^A, f^A, \mathcal{L}_A \rangle$ and $\langle r+1, H_B, U \setminus A, T^B, f^B, \mathcal{L}_B \rangle$.

Correctness. There are two properties to check in order to prove the validity of our approach. The first such property is that, being given the two gadget subgraphs H_A and H_B resulting from H, the distances in H (and so, in G) are preserved. This follows from Lemma 15. The second property to be checked is that we always compute a cut $(A', V(H) \setminus A')$ of neighbourhood diversity at most k. We prove by induction on r, using Lemma 16, that:

▶ **Property 1.** *For every $\langle r, H, U, T^U, f^U, \mathcal{L} \rangle$, let $s_1, s_2, \ldots, s_q$ be children of some node s in T^U. Let S_i be the subset of U of which $f^U(s_i)$ is a partition, and set $S = \bigcup_{i=1}^{q} S_i$. Finally, let S' be the union of S with all subsets W_j, for $(W_j, c_j) \in \mathcal{L}$, s.t. the least common ancestor in T of c_j and some node s_i is a strict descendant of s. Then, S' is a k-module of H.*

Complexity analysis. By induction, the depth of the recursion tree is $\mathcal{O}(\log n)$. Furthermore, the sum of all the orders $|V(H)| + |E(H)|$, over all the inputs $\langle r, H, U, T^U, f^U, \mathcal{L} \rangle$ that are at the same recursion level r, is at most $\mathcal{O}(k^2 n \log n + m)$. Therefore, by Lemmas 14 and 12 the total running time at any fixed recursive stage, and so also for the whole algorithm, is in $\mathcal{O}(2^{\mathcal{O}(k)}(n+m)^{1+\epsilon})$, for any $\epsilon > 0$. ◀

5 Facility location problems on bounded clique-width graphs

Our last result in the paper is as follows:

▶ **Theorem 17.** *For every connected n-vertex m-edge graph $G = (V, E)$, if $cw(G) \leq k$ and a k-expression is given, then for any $\epsilon > 0$, we can compute in $\mathcal{O}(2^{\mathcal{O}(k)}(n+m)^{1+\epsilon})$ time: all the p-eccentricities and all the total p-distances, for every cost function $p : V \to \mathbb{N}$.*

Despite its apparent similarity with Theorem 8, Theorem 17 has some special features. To see why, let us assume that two vertices u, v are disconnected by a join with respective sides X, Y. Then, $d(u, v) = d(u, X) + 1 + d(Y, v)$,[2] and therefore for any fixed u, in order to maximize $d(u, v)$ it suffices to find such a v maximizing $d(v, Y)$. However, this is no more true if we have a cost function p; indeed, we now want to maximize $p(v) \cdot (d(u, X)+1)+p(v)d(v, Y)$.

For that, we first prove that:

▶ **Lemma 18.** *Let F be a set of n linear functions $f_i : t \to a_i \cdot t + b_i$, where $a_i, b_i \geq 0$. Then after an $\mathcal{O}(n \log n)$-time pre-processing, for any $x \geq 0$ we can compute $\max_{1 \leq i \leq n}\{a_i \cdot x + b_i\}$ in $\mathcal{O}(\log n)$ time.*

Combined with some insights of Cabello about range trees [12], we get:

▶ **Corollary 19.** *Let P be a set of n points in $\mathbb{R}^d$ where each point $p \in P$ is associated an ordered pair $(a(p), b(p))$ of nonnegative real numbers. We can construct a data structure in $\mathcal{O}(2^{\mathcal{O}(d)}n^{1+\epsilon})$ time, for any $\epsilon > 0$, such that, for any box $\mathcal{R}$ and nonnegative $x \geq 0$, a point $p \in P \cap \mathcal{R}$ maximizing $a(p) \cdot x + b(p)$ can be output in $\mathcal{O}(2^{\mathcal{O}(d)}n^{\epsilon})$ time.*

▶ **Lemma 20.** *Let $G = (V, E, \alpha)$ be a connected n-vertex m-edge graph, where $\alpha : E \to \mathbb{N}$, and let $p \geq 0$ be some vertex-weight function. Let also $(A, V \setminus A)$ be an unweighted cut of neighbourhood diversity at most k, and let $A' \subseteq A$, $B' \subseteq V \setminus A$. For any $\epsilon > 0$, after a pre-processing in $\mathcal{O}(km + 2^{\mathcal{O}(k)}n^{1+\epsilon})$ time, for every vertex $u \in A'$ we can compute the values $\max_{v \in B'} p(v) \cdot d_G(u, v)$ and $\sum_{v \in B'} p(v) \cdot d_G(u, v)$ in $\mathcal{O}(2^{\mathcal{O}(k)}n^{\epsilon})$ time; in the same way, for every vertex $v \in B'$ we can compute the values $\max_{u \in A'} p(u) \cdot d_G(v, u)$ and $\sum_{u \in A'} p(u) \cdot d_G(v, u)$ in $\mathcal{O}(2^{\mathcal{O}(k)}n^{\epsilon})$ time.*

Theorem 17 now follows from the exact same proof as for Theorem 8, but where we use Lemma 20 rather than Lemma 12.

References

1. Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 377–391. SIAM, 2016. doi:10.1137/1.9781611974331.ch28.
2. John Adrian Bondy and Uppaluri Siva Ramachandra Murty. *Graph theory*, volume 244 of *Graduate Texts in Mathematics*. Springer-Verlag London, 2008.
3. R Borie, JL Johnson, V Raghavan, and JP Spinrad. Robust polynomial time algorithms on clique-width k graphs. 2002.
4. Andreas Brandstädt, Konrad K Dabrowski, Shenwei Huang, and Daniël Paulusma. Bounding the clique-width of H-free split graphs. *Discrete Applied Mathematics*, 211:30–39, 2016.
5. Andreas Brandstädt, Konrad K Dabrowski, Shenwei Huang, and Daniël Paulusma. Bounding the Clique-Width of H-Free Chordal Graphs. *Journal of Graph Theory*, 86(1):42–77, 2017.
6. Andreas Brandstädt, Feodor F Dragan, Hoàng-Oanh Le, and Raffaele Mosca. New graph classes of bounded clique-width. *Theory of Computing Systems*, 38(5):623–645, 2005.

[2] This is a slightly different formula than in Lemma 5, which is for vertex-weighted graphs. Here we adapt the formula for unweighted graphs, where the distance between two vertices u and v is classically defined as the minimum number of edges on a uv-path.

7 Andreas Brandstadt, Joost Engelfriet, Hoang-Oanh Le, and Vadim V Lozin. Clique-width for 4-vertex forbidden subgraphs. *Theory of Computing Systems*, 39(4):561–590, 2006.

8 Andreas Brandstädt, Tilo Klembt, and Suhail Mahfud. P_6-and triangle-free graphs revisited: structure and bounded clique-width. *Discrete Mathematics & Theoretical Computer Science*, 8(1), 2006.

9 Andreas Brandstädt, Hoàng-Oanh Le, and Raffaele Mosca. Gem-and co-gem-free graphs have bounded clique-width. *International Journal of Foundations of Computer Science*, 15(01):163–185, 2004.

10 Andreas Brandstädt, Hoàng-Oanh Le, and Raffaele Mosca. Chordal co-gem-free and (P_5, gem)-free graphs have bounded clique-width. *Discrete Applied Mathematics*, 145(2):232–241, 2005.

11 K. Bringmann, T. Husfeldt, and M. Magnusson. Multivariate Analysis of Orthogonal Range Searching and Graph Distances. *Algorithmica*, pages 1–24, 2020.

12 S. Cabello. Computing the inverse geodesic length in planar graphs and graphs of bounded treewidth. Technical Report 1908.01317, arXiv, 2019.

13 S. Cabello and C. Knauer. Algorithms for graphs of bounded treewidth via orthogonal range searching. *Computational Geometry*, 42(9):815–824, 2009.

14 Derek G. Corneil, Michel Habib, Jean-Marc Lanlignel, Bruce Reed, and Udi Rotics. Polynomial Time Recognition of Clique-Width ≤ 3 Graphs. In *Latin American Theoretical INformatics Symposium (LATIN)*, volume 1776 of *Lecture Notes in Computer Science*, pages 126–134. Springer, 2000. doi:10.1007/10719839_14.

15 Derek G. Corneil and Udi Rotics. On the relationship between clique-width and treewidth. *SIAM Journal on Computing*, 34(4):825–847, 2005. doi:10.1137/S0097539701385351.

16 D. Coudert, G. Ducoffe, and A. Popa. Fully polynomial FPT algorithms for some classes of bounded clique-width graphs. *ACM Transactions on Algorithms (TALG)*, 15(3):1–57, 2019.

17 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

18 Bruno Courcelle, Pinar Heggernes, Daniel Meister, Charis Papadopoulos, and Udi Rotics. A characterisation of clique-width through nested partitions. *Discrete Applied Mathematics*, 187:70–81, 2015. doi:10.1016/j.dam.2015.02.016.

19 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33(2):125–150, 2000. doi:10.1007/s002249910009.

20 Bruno Courcelle and Rémi Vanicat. Query efficient implementation of graphs of bounded clique-width. *Discrete Applied Mathematics*, 131(1):129–150, 2003.

21 William H. Cunningham. Decomposition of directed graphs. *SIAM Journal on Algebraic Discrete Methods*, 3(2):214–228, 1982. doi:10.1137/0603021.

22 Konrad K Dabrowski and Daniël Paulusma. Classifying the clique-width of H-free bipartite graphs. *Discrete Applied Mathematics*, 200:43–51, 2016.

23 Konrad K Dabrowski and Daniël Paulusma. Clique-width of graph classes defined by two forbidden induced subgraphs. *The Computer Journal*, 59(5):650–666, 2016.

24 K. Das, S. Samanta, and M. Pal. Study on centrality measures in social networks: a survey. *Social network analysis and mining*, 8(1):1–11, 2018.

25 Reinhard Diestel. *Graph Theory*. Graduate Texts in Mathematics. Springer, 2010. 4*th* edition. doi:10.1007/978-3-662-53622-3.

26 Feodor F Dragan and Chenyu Yan. Collective tree spanners in graphs with bounded parameters. *Algorithmica*, 57(1):22–43, 2010.

27 Guillaume Ducoffe and Alexandru Popa. The b-matching problem in distance-hereditary graphs and beyond. In *International Symposium on Algorithms and Computation (ISAAC)*, volume 123 of *Leibniz International Proceedings in Informatics*, pages 30:1–30:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.ISAAC.2018.30.

28 Guillaume Ducoffe and Alexandru Popa. The use of a pruned modular decomposition for maximum matching algorithms on some graph classes. In *International Symposium on Algorithms and Computation (ISAAC)*, volume 123 of *Leibniz International Proceedings in Informatics*, pages 6:1–6:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.ISAAC.2018.6.

29 Wolfgang Espelage, Frank Gurski, and Egon Wanke. How to solve NP-hard graph problems on clique-width bounded graphs in polynomial time. In *International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, volume 1 of *Lecture Notes in Computer Science*, pages 117–128. Springer, 2001. doi:10.1007/3-540-45477-2_12.

30 Michael R. Fellows, Frances A. Rosamond, Udi Rotics, and Stefan Szeider. Clique-width is NP-complete. *SIAM Journal on Discrete Mathematics*, 23(2):909–939, 2009. doi:10.1137/070687256.

31 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of clique-width parameterizations. *SIAM Journal on Computing*, 39(5):1941–1956, 2010. doi:10.1137/080742270.

32 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Almost optimal lower bounds for problems parameterized by clique-width. *SIAM Journal on Computing*, 43(5):1541–1563, 2014. doi:10.1137/130910932.

33 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Clique-width III: Hamiltonian Cycle and the Odd Case of Graph Coloring. *ACM Transactions on Algorithms*, 15(1):9, 2019. doi:10.1145/3280824.

34 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, Michal Pilipczuk, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. *ACM Transactions on Algorithms*, 14(3):34:1–34:45, 2018. doi:10.1145/3186898.

35 Martin Fürer. A natural generalization of bounded tree-width and bounded clique-width. In *Latin American Symposium on Theoretical Informatics*, pages 72–83. Springer, 2014.

36 C. Gavoille, D. Peleg, S. Pérennes, and R. Raz. Distance labeling in graphs. *Journal of Algorithms*, 53(1):85–112, 2004.

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

38 Archontia C. Giannopoulou, George B. Mertzios, and Rolf Niedermeier. Polynomial fixed-parameter algorithms: A case study for longest path on interval graphs. *Theoretical computer science*, 689:67–95, 2017. doi:10.1016/j.tcs.2017.05.017.

39 A. Goldman. Optimal center location in simple networks. *Transportation science*, 5(2):212–221, 1971.

40 Martin Charles Golumbic and Udi Rotics. On the clique-width of some perfect graph classes. *International Journal of Foundations of Computer Science*, 11(03):423–443, 2000. doi:10.1142/S0129054100000260.

41 P. Hage and F. Harary. Eccentricity and centrality in networks. *Social networks*, 17(1):57–63, 1995.

42 Torben Hagerup, Jyrki Katajainen, Naomi Nishimura, and Prabhakar Ragde. Characterizing multiterminal flow networks and computing flows in networks of small treewidth. *Journal of Computer and System Sciences*, 57(3):366–375, 1998.

43 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

44 Yoichi Iwata, Tomoaki Ogasawara, and Naoto Ohsaka. On the power of tree-depth for fully polynomial FPT algorithms. In *International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 96 of *Leibniz International Proceedings in Informatics*, pages 41:1–41:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.STACS.2018.41.

45 Shahin Kamali. Compact representation of graphs of small clique-width. *Algorithmica*, 80(7):2106–2131, 2018.

46 Stefan Kratsch and Florian Nelles. Efficient and adaptive parameterized algorithms on modular decompositions. In *European Symposia on Algorithms (ESA)*, pages 55:1–55:15, 2018. doi:10.4230/LIPIcs.ESA.2018.55.

47 Stefan Kratsch and Florian Nelles. Efficient Parameterized Algorithms for Computing All-Pairs Shortest Paths. In *37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020)*, volume 154 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 38:1–38:15, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2020.38.

48 V Lozin and Dieter Rautenbach. Chordal bipartite graphs of bounded tree-and clique-width. *Discrete Mathematics*, 283(1-3):151–158, 2004.

49 Johann A. Makowsky and Udi Rotics. On the clique-width of graphs with few P_4's. *International Journal of Foundations of Computer Science*, 10(03):329–348, 1999. doi:10.1142/S0129054199000241.

50 S. Oum and P. Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B*, 96(4):514–528, 2006.

51 Michaël Rao. Clique-width of graphs defined by one-vertex extensions. *Discrete Mathematics*, 308(24):6157–6165, 2008. doi:10.1016/j.disc.2007.11.039.

52 G. Sabidussi. The centrality index of a graph. *Psychometrika*, 31(4):581–603, 1966.

53 Karol Suchan and Ioan Todinca. On powers of graphs of bounded NLC-width (clique-width). *Discrete Applied Mathematics*, 155(14):1885–1893, 2007.

54 Jean-Marie Vanherpe. Clique-width of partner-limited graphs. *Discrete mathematics*, 276(1-3):363–374, 2004.

Polynomial Kernels for Strictly Chordal Edge Modification Problems

Maël Dumas
Univ. Orléans, INSA Centre Val de Loire, LIFO EA 4022, F-45067 Orléans, France

Anthony Perez
Univ. Orléans, INSA Centre Val de Loire, LIFO EA 4022, F-45067 Orléans, France

Ioan Todinca
Univ. Orléans, INSA Centre Val de Loire, LIFO EA 4022, F-45067 Orléans, France

Abstract

We consider the Strictly Chordal Editing problem, where one is given an undirected graph $G = (V, E)$ and a parameter $k \in \mathbb{N}$ and seeks to *edit* (add or delete) at most k edges from G to obtain a strictly chordal graph. Problems Strictly Chordal Completion and Strictly Chordal Deletion are defined similarly, by only allowing edge additions for the former, and only edge deletions for the latter. Strictly chordal graphs are a generalization of 3-leaf power graphs and a subclass of 4-leaf power graphs. They can be defined, e.g., as dart and gem-free chordal graphs. We prove the NP-completeness for all three variants of the problem and provide an $O(k^3)$ vertex-kernel for the completion version and $O(k^4)$ vertex-kernels for the two others.

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

Keywords and phrases Parameterized complexity, kernelization algorithms, graph modification, strictly chordal graphs

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.17

1 Introduction

Parameterized algorithms are among the most natural approaches to tackle NP-hard optimization problems [13]. In particular, they have been very successful in dealing with so-called edge modification problems on graphs: given as input an arbitrary graph $G = (V, E)$ and a parameter $k \in \mathbb{N}$, the goal is to transform G into a graph with some specific properties (i.e., belonging to a specific graph class $\mathcal{G}$) by adding and/or deleting at most k edges. Parameterized algorithms (also called FPT for *fixed parameter tractable*) aim at a time complexity of type $f(k) \cdot n^{O(1)}$, where f is some computable function, hence the combinatorial explosion is restricted to parameter k.

When the target class $\mathcal{G}$ is characterized by a finite family of forbidden induced subgraphs, modification problems are FPT by a result of Cai [8]. Indeed, as long as our graph contains one of the forbidden subgraphs, we can try each possibility to correct this obstruction and branch by recursive calls. On each branch, the budget k is strictly diminished, therefore the whole algorithm has a number of calls bounded by some function $f(k)$. The situation is more complicated when the target class $\mathcal{G}$ is characterized by an infinite family of forbidden induced subgraphs. Nonetheless, a large literature is devoted to edge modification problems towards chordal graphs (where we forbid all induced cycles with at least four vertices) as well as sub-classes of chordal graphs, typically obtained by requiring some fixed set of obstructions, besides the long cycles. Observe that, in this case, the situation remains relatively simple if we restrict ourselves to *edge completion* problems, where we are only allowed to *add* edges to the input graphs. Indeed, in this case, if a graph has a cycle of length longer than $k + 3$, it cannot be made chordal by adding at most k edges. Therefore we can use again the

approach of Cai to deal with cycles of length at most $k+3$ and other obstructions, and either the algorithm finds a solution in $f(k)$ recursive calls, or we can conclude that we have a no-instance. The cases of *edge deletion* problems (where we are only allowed to remove edges) and *edge editing* problems (where we are allowed to both remove edges or add missing edges) are more complicated, since even long cycles can be eliminated by a single edge removal. Therefore more efforts and more sophisticated techniques were necessary in these situations, but several such problems turned out to be FPT [10, 14, 15]. The interested reader can refer to [11] for a broad and comprehensive survey on parameterized algorithms for edge modification problems.

We focus here on a sub-family of parameterized algorithms, namely on *kernelization*. The goal of kernelization is to provide a polynomial algorithm transforming any instance (I, k) of the problem into an equivalent instance (I', k') where k' is upper bounded by some function of k (in our case we will simply have $k' \leq k$), and the size of the new instance I' is upper bounded by some function $g(k)$. Hence the size of the reduced instance does not depend on the size of the original instance. While kernelization is possible for all FPT problems (the two notions are actually equivalent), the interesting question is whether a given FPT problem admits *polynomial kernels*, where the size of the reduced instance is bounded by some polynomial in k. Note that, under some complexity assumptions, not all FPT problems admit polynomial kernels [5, 6, 7, 9, 19, 24].

In this paper we provide polynomial kernels for Strictly Chordal Completion, Strictly Chordal Deletion and Strictly Chordal Editing. *Strictly chordal* graphs are a subclass of chordal graphs, also known as *block duplicate* graphs [22, 23, 18]. They can be obtained from block graphs, i.e., graphs in which every block (bi-connected component) induces a clique, by repeatedly choosing some vertex u and adding a *true twin* v of u, that is a vertex v adjacent to u and all other neighbors of u. They can also be characterized as dart, gem-free chordal graphs (see Figure 1 and next section). Strictly chordal graphs are known to be a subclass of 4-leaf power graphs [23], and a super-class of 3-leaf power graphs. Leaf power graphs have been introduced in the context of phylogeny reconstruction [28]. A graph is said to be p-leaf power, for some integer p, if its vertices can be bijectively mapped onto the leaves of some tree, such that two vertices are adjacent in the graph if the corresponding leaves are at distance at most p in the tree.

Related work

Kernelization for chordal completion goes back to the '90s and the seminal paper of Kaplan, Shamir and Tarjan [21]. Since then, several authors addressed completion, deletion and/or editing problems towards sub-classes of chordal graphs, as 3-leaf power graphs [3], split and threshold graphs [20], proper interval graphs [4], trivially perfect graphs [2, 16, 17, 20] or ptolemaic graphs [12]. All these classes have in common that they can be defined as chordal graphs, plus a constant number of obstructions. Several questions remain open, for example it is not known whether chordal deletion or chordal editing admit polynomial kernels [11]. We could also ask whether completion and editing problems towards 4-leaf power graphs admit a polynomial kernel, knowing that they are FPT [15].

Our contribution

Firstly, we prove that problems Strictly Chordal Completion, Strictly Chordal Deletion and Strictly Chordal Editing are NP-complete. Secondly, we give a kernelization algorithm for the Strictly Chordal Completion problem, producing a reduced

instance with $O(k^3)$ vertices. Eventually, we discuss how to extend this approach in order to obtain an $O(k^4)$-vertex kernel for both Strictly Chordal Deletion and Strictly Chordal Editing. Above all, our purpose is to exhibit general techniques that might, we hope, be extended to kernelizations for edge modification problems towards other graph classes. Several such algorithms, e.g., [3, 17] share the following feature. Very informally, the target class $\mathcal{G}$ admits a tree-like decomposition, in the sense that the vertices of any graph $H \in \mathcal{G}$ can be partitioned into clique modules, and these modules can be mapped onto the nodes of a decomposition tree, the structure of the tree describing the adjacencies between modules. Therefore, if an arbitrary graph G can be transformed into graph H by at most k edge additions or deletions, at most $2k$ modules can be affected by the modifications. By removing the affected nodes from the decomposition tree, we are left with several components that correspond, in the initial graph G as well as in H, to induced subgraphs that may be large but that already belong to the target class. Moreover, these chunks are attached to the rest of graph G in a very regular way, through one or two nodes of the decomposition tree. The kernelization algorithms need to analyze these chunks and provide reduction rules, typically by ensuring a small number of nodes in the decomposition tree, plus the fact that each node corresponds to a module of small size.

The class of strictly chordal graphs does not have exactly a tree-like decomposition, but still can be decomposed into a structure of *block graph*, which can be seen as a generalization of a tree. Our algorithms exploit these informal observations and provide the necessary reduction rules together with the combinatorial analysis for the kernel size. Proofs of statements labeled with ($\star$) are omitted in this extended abstract.

2 Preliminaries

We consider simple, undirected graphs $G = (V, E)$ where V denotes the *vertex set* and $E \subseteq (V \times V)$ the *edge set* of G. We will sometimes use $V(G)$ and $E(G)$ to clarify the context. Given a vertex $u \in V$, the *open neighborhood* of u is the set $N_G(u) = \{v \in V : uv \in E\}$. The *closed neighborhood* of u is defined as $N_G[u] = N_G(u) \cup \{u\}$. Two vertices u and v are *true twins* if $N_G[u] = N_G[v]$. Given a subset of vertices $S \subseteq V$, $N_G[S]$ is the set $\cup_{v \in S} N_G[v]$ and $N_G(S)$ is the set $N_G[S] \setminus S$. We will omit the mention to G whenever the context is clear. The subgraph *induced* by S is defined as $G[S] = (S, E_S)$ where $E_S = \{uv \in E : u \in S, v \in S\}$. For the sake of readability, given a subset $S \subseteq V$ we define $G \setminus S$ as $G[V \setminus S]$. A subgraph C is a *connected component* of G if it is a maximal connected subgraph of G. A subset of vertices $M \subseteq V$ is a *module* if for every vertices $x, y \in M$, $N(x) \setminus M = N(y) \setminus M$. A set $S \subseteq V$ is a *separator* of G if $G \setminus S$ is not connected. Given two vertices u and v of G, the separator S is a uv-separator if u and v lie in distinct connected components of $G \setminus S$. Moreover, S is a *minimal* uv-separator if no proper subset of S is a uv-separator. Finally, a separator S is *minimal* if there exists a pair $\{u, v\}$ such that S is a minimal uv-separator.

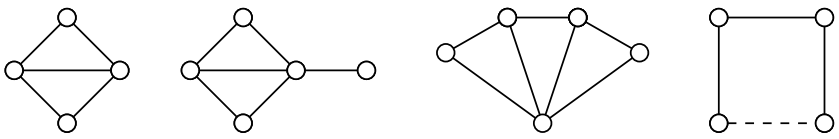


Figure 1 The diamond, dart, gem and cycle of length at least 4.

▶ **Definition 1.** *Given a graph $G = (V, E)$, a* critical clique *of G is a set $K \subseteq V$ such that $G[K]$ is a clique, K is a module and is inclusion-wise maximal under this property.*

Notice that K is a maximal set of true twins and that the set $\mathcal{K}(G)$ of critical cliques of any graph G partitions its vertex set $V(G)$. This leads to the following definition.

▶ **Definition 2** (Critical clique graph). *Let $G = (V, E)$ be a graph. The* critical clique graph *of G is the graph $\mathcal{C}(G) = (\mathcal{K}(G), E_{\mathcal{C}})$ with $E_{\mathcal{C}} = \{KK' \mid \forall u \in K, \forall v \in K', uv \in E\}$.*

Strictly chordal graphs

Block graphs are graphs in which every block (bi-connected component) is a clique. They can also be characterized as chordal graphs that do not contain diamonds as induced subgraphs [1]. A natural generalization of block graphs are strictly chordal graphs, also known as block duplicate graphs [22, 23, 18], that are obtained from block graphs by adding true twins [18].

▶ **Theorem 3** ([26]). *Let $G = (V, E)$ be a graph. The following conditions are equivalent:*

1. *G is a strictly chordal graph,*
2. *The critical clique graph $\mathcal{C}(G)$ is a block graph,*
3. *G does not contain any dart, gem or hole as an induced subgraph (see Figure 1),*
4. *The minimal separators of G are pairwise disjoint.*

We consider the following problem:

Strictly Chordal Editing
Input: A graph $G = (V, E)$, a parameter $k \in \mathbb{N}$
Question: Does there exist a set of pairs $F \subseteq (V \times V)$ of size at most k such that the graph $H = (V, E \triangle F)$ is strictly chordal, with $E \triangle F = (E \setminus F) \cup (F \setminus E)$?

The Strictly Chordal Completion and Strictly Chordal Deletion problems are defined similarly by requiring F to be disjoint from (resp. included in) edge set E. Given a graph $G = (V, E)$, a set $F \subseteq (V \times V)$ such that the graph $H = (V, E \triangle F)$ is strictly chordal is called an *edition* of G. When F is disjoint from E (resp. included in E) it is called a *completion* (resp. a *deletion*) of G. For the sake of simplicity we use $G \triangle F$, $G + F$ and $G - F$ to denote the resulting strictly chordal graphs in all version of the problem. In all cases, F is *optimal* whenever it is minimum-sized. Given an instance $(G = (V, E), k)$ of any of the aforementioned problems, we say that F is a *k-edition* (resp. k-completion, k-deletion) whenever F is an edition (resp. completion, deletion) of size at most k. Finally, a vertex is *affected* by F whenever it is contained in some pair of F. We say that an instance $(G = (V, E), k)$ of any of the aforementioned problems is a yes-instance whenever it admits a k-edition (resp. k-completion, k-deletion).

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two vertex-disjoint graphs and let $S_1 \subseteq V_1, S_2 \subseteq V_2$. The *join composition* of G_1 and G_2 on S_1 and S_2, denoted $(G_1, S_1) \otimes (G_2, S_2)$, is the graph $(V_1 \cup V_2, E_1 \cup E_2 \cup (S_1 \times S_2))$.

▶ **Lemma 4** ([22]). *Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two strictly chordal graphs and let $S_1 \subseteq V_1, S_2 \subseteq V_2$. The graph $G = (G_1, S_1) \otimes (G_2, S_2)$ is strictly chordal if for each $i \in \{1, 2\}$, S_i is a critical clique or is a clique included in exactly one maximal clique of G_i.*

We will use the following result that guarantees that any clique module of a given graph G will remain a clique module in any optimal edge modification towards some hereditary class of graphs, in particular towards strictly chordal graphs.

▶ **Lemma 5** ([3]). *Let $\mathcal{G}$ be an hereditary class of graphs closed under true twin addition. For every graph $G = (V, E)$, there exists an optimal edition (resp. completion, deletion) F into a graph of $\mathcal{G}$ such that for any two critical cliques K and K' either $(K \times K') \subseteq F$ or $(K \times K') \cap F = \emptyset$.*

In the remainder of this paper we always assume that the considered optimal editions (resp. completions, deletions) satisfy Lemma 5.

2.1 Hardness results

The NP-completeness of Strictly Chordal Completion follows directly from the proof of NP-completeness of 3-Leaf Power Completion from [14]. We can use the same reduction from Biclique Deletion, taking the complement of a bipartite graph and adding an universal vertex.

▶ **Theorem 6.** *Strictly Chordal Completion is NP-complete.*

We show the NP-completeness of Strictly Chordal Editing and Strictly Chordal Deletion by giving a reduction from Cluster Editing and Cluster Deletion, known to be NP-complete [25, 30, 27].

▶ **Theorem 7.** *Strictly Chordal Editing and Strictly Chordal Deletion are NP-complete.*

Proof. A graph is a cluster graph if it does not contain any induced path on three vertices (so-called P_3). Given an instance $(G = (V, E), k)$ of Cluster Editing, we construct an instance of Strictly Chordal Editing by adding a clique $U = \{u_1, \ldots, u_{k+1}\}$ of size $k + 1$ adjacent to all vertices of V, and for each vertex x in V, $k + 1$ vertices $\{v_1^x, \ldots, v_{k+1}^x\}$ adjacent only to x. Let $(G' = (V', E'), k)$ be the produced instance. We show that the graph G admits a k-edition into a cluster graph if and only if G' admits a k-edition into a strictly chordal graph. Suppose first that there is a k-edition F of G into a cluster graph. The graph $G \triangle F$ is a graph without any P_3 as induced subgraph. Now consider the graph $H' = G' \triangle F$. By construction $H'[V]$ contains no P_3, so H' is chordal and contains neither gems nor dart. By Theorem 3, it follows that H' is strictly chordal. Now suppose that there exists a k-edition F' of G' into a strictly chordal graph. By contradiction, suppose that G' contains a P_3 $\{x, y, z\}$ where x, z are the ends of the path. Then, there exist $i, j \in \{1, \ldots, k + 1\}$ such that $\{x, y, z, u_i, v_j^y\}$ forms a dart, contradicting that $G' \triangle F'$ is strictly chordal, thus $G \triangle F$ is a cluster graph.

The same reduction can be done from Cluster Deletion to Strictly Chordal Deletion. This concludes the proof. ◀

3 Kernelization algorithm for Strictly Chordal Completion

We begin this section by providing a high-level description of our kernelization algorithm. We use the critical clique graph of strictly chordal graphs to bound the number of vertices of a reduced instance. Let us consider a positive instance $(G = (V, E), k)$ of Strictly Chordal Completion, F a suitable solution and $H = G + F$. Denote by $\mathcal{C}(H)$ the critical clique graph of H as described Definition 2 and recall that $\mathcal{C}(H)$ is a block graph (Theorem 3). Since $|F| \leq k$, we know that at most $2k$ critical cliques of $\mathcal{C}(H)$ may contain affected vertices. Let A be the set of such critical cliques, T the minimum subgraph of $\mathcal{C}(H)$ that spans all critical cliques of A and A' the set of critical cliques of degree at least 3 in T. We shall see later that $|A' \backslash A| \leq 3 \cdot |A|$ (Lemma 19). We will define the notion of block-branch, corresponding to subgraphs of G that are strictly chordal. We will focus our interest on two types of block-branches: the ones that are connected to the rest of the graph by only one critical clique, called *1-block-branches*, and the ones that are connected to the rest of the graph by exactly two critical cliques, called *2-block-branches*. The connected components

of the graph $T\backslash(A \cup A')$ correspond to parts of 2-block-branches and the length of these 2-block-branches will be bounded by 3. We will see that there are at most $4k$ such connected components, thus there is $O(k)$ critical cliques in T. Finally, the connected components of the graph $\mathcal{C}(H)\backslash V(T)$ correspond to 1-block-branches or sets of connected 1-block-branches. Each 1-block-branch will be reduced to 2 critical cliques, and to each critical clique or maximal clique of T there is a linear number of 1-block-branches of $\mathcal{C}(H)\backslash V(T)$ adjacent to it. Altogether, the graph $\mathcal{C}(H)$ contains $O(k^2)$ critical cliques, and each critical clique is of size at most $k+1$, hence the graph G contains $O(k^3)$ vertices.

3.1 Classical reduction rules

We first give classical reduction rules when dealing with modification problems. The first rule is safe for any target graph class hereditary and closed under disjoint union. The second one comes from the fact that strictly chordal graphs are hereditary and closed under true twin addition, combined with Lemma 5.

▶ **Rule 1.** *Let C be a strictly chordal connected component of G. Remove $V(C)$ from G.*

▶ **Rule 2.** *Let $K \subseteq V$ be a set of true twins of G such that $|K| > k+1$. Remove $|K|-(k+1)$ arbitrary vertices in K from G.*

▶ **Lemma 8** (Folklore, [3])**.** *Rules 1 and 2 are safe and can be applied in polynomial time.*

3.2 Block-branch reduction rules

We now consider the main structure of our kernelization algorithm, namely *block-branches*.

▶ **Definition 9** (block-branch)**.** *Let $G = (V, E)$ be a graph. We say that a subgraph B of G is a* block-branch *if $V(B)$ is an union of critical cliques $K_1, \ldots, K_r \in \mathcal{K}(G)$ such that the subgraph of $\mathcal{C}(G)$ induced by $K_1, \ldots, K_r$ is a connected block graph.*

We emphasize that our definition of a block-branch B is stronger than simply requiring B to be an induced strictly chordal graph. For example, if G is the dart graph, the subgraph obtained by removing the pendant vertex is strictly chordal, but it is not a block-branch because the corresponding critical cliques do not form a block graph in $\mathcal{C}(G)$. Let B be a block-branch of graph G and let $K_1, \ldots, K_r$ be the critical cliques of G contained in $V(B)$. We say that K_i is an *attachment point* of B if $N_G(K_i) \setminus V(B) \neq \emptyset$. A block-branch B is a p-block-branch if it has exactly p attachment points. We denote B^R the subgraph of B in which all attachment points have been removed.

We first give structural Lemmata on block-branches that will be helpful to prove the safeness of our rules.

▶ **Lemma 10.** *Let $G = (V, E)$ be a graph and B a block-branch of G. For any attachment point P of B, let $B' = B\backslash P$, consider the connected components $G_1, G_2, \ldots, G_r$ of B' and let $Q_i = N_B(P) \cap V(G_i)$. For every i, $1 \leq i \leq r$, Q_i is a critical clique of G_i or Q_i is included in exactly one maximal clique of G_i.*

Proof. First, we show that all sets Q_i are cliques. Suppose that Q_i is not a clique for some $1 \leq i \leq r$. Let x and y be non adjacent vertices of Q_i and $z \in P$. Since G_i is connected, take a shortest path π between x and y in G_i. The subgraph induced by the vertices $\{x, y, z\}$ and those of π contains either a cycle of length at least 4 if z is not adjacent to any inner vertex of π, which is a contradiction since B is a block graph, or a diamond with z being

one of its vertices of degree 3. In the latter case, since z is not in the same critical clique of $\mathcal{C}(G)$ as its true twin in the diamond, the critical cliques of $\mathcal{C}(G)$ that contain some vertices of this diamond also form a diamond in $\mathcal{C}(G)$. This diamond is formed by critical cliques of G contained in B, contradicting the definition of a block-branch. In all cases we have a contradiction, it remains that Q_i is a clique in G.

Now, suppose that Q_i, $1 \leq i \leq r$ is included in two or more maximal cliques in G_i and let $u, v \in V(G_i) \backslash Q_i$ be two vertices adjacent to Q_i such that $uv \notin E(G_i)$. If Q_i is not a critical clique of G_i, there are two cases, either Q_i is not a module of G_i or Q_i is included in a larger module of G_i. In the first case, there are two vertices $x, y \in Q_i$ that are not in the same module of G_i. Let $z \in V(G_i) \backslash Q_i$ be a vertex adjacent to only x or y, say w.l.o.g. x. The subgraph of G induced by $\{u, v, x, y, z\}$ is either a dart if z is not adjacent to u nor v or a gem if it is adjacent to exactly one of them. If z is adjacent to both u and v, then $\{z, u, y, v\}$ forms a C_4 in this order, leading to a contradiction. In the second case, that is Q_i is included in a larger module of G_i, let $x \in Q_i$, y a vertex in the same module as x in G_i and $z \in P$. Observe that $zy \notin E(G)$. The set $\{x, y, z, u, v\}$ induces a dart in G. In all cases we have a contradiction, since the forbidden structure is also contained in B. It remains that Q_i is a critical clique of G_i or Q_i is included in exactly one maximal clique of G_i. ◀

▶ **Lemma 11.** *Let $G = (V, E)$ be a graph and B a block-branch of G. Let F be an optimal completion of G that respects Lemma 5, and $H = G + F$. For any attachment point P of B, let C be the critical clique of H which contains P. Then $C' = C \backslash V(B^R)$ is a critical clique of $H' = H \backslash V(B^R)$ or is included in exactly one maximal clique of H'.*

Proof. Assume that C' is included in two or more maximal cliques in H'. Let $u, v \in V(H') \backslash C'$ be two vertices adjacent to C' such that $uv \notin E(H')$. If C' is not a critical clique of H', then either C' is not a module of H' or C' is included in a larger module of H'. In the first case, there are two vertices $x, y \in C'$ that are not in the same module of H'. Let $z \in V(H') \backslash C'$ be a vertex adjacent to only x or y, say w.l.o.g. x. The subgraph of H induced by $\{u, v, x, y, z\}$ is either a dart if z is not adjacent to u nor v or a gem if it is adjacent to exactly one of them. If z is adjacent to both u and v, then $\{z, u, y, v\}$ forms a C_4 in this order, leading to a contradiction. In the second case, C' is included in a larger module of H'. Let $x \in C'$, y be a vertex in the same module as x in H' and $z \in N_B(P)$. The set $\{x, y, z, u, v\}$ induces a dart in H if zy is not in $E(H)$, else a gem in H. In all cases we have a contradiction since H is strictly chordal. It remains that C' is a critical clique of H' or C' is included in exactly one maximal clique of H'. ◀

▶ **Lemma 12.** *Let $G = (V, E)$ be a graph and B a 1-block-branch of G with attachment point P. There exists an optimal completion F of G such that:*

- *The set of vertices of B affected by F is included in $P \cup N_B(P)$.*
- *In $H = G + F$ the vertices of $N_B(P)$ are all adjacent to the same vertices of $V(G) \backslash V(B^R)$.*

Proof. Let F be an optimal completion of G. Let C be the critical clique of H which contains P and let $C' = C \backslash V(B^R)$. Let $Q_1, \ldots, Q_r$ be the cliques that partition $N_B(P)$ (Lemma 10) and G_i the connected component of B^R which contains Q_i. The graphs $H' = H \backslash V(B^R)$ and G_i, for $1 \leq i \leq r$, are strictly chordal by heredity. By Lemma 10, for $1 \leq i \leq r$, Q_i is a critical clique of G_i or is in exactly one maximal clique of G_i. By Lemma 11, C' is a critical clique of H' or is in exactly one maximal clique of H'. Thus, by Lemma 4 the graph H^* corresponding to the graph $(\bigcup_{1 \leq i \leq r} G_i, \bigcup_{1 \leq i \leq r} Q_i) \otimes (H', C')$ is a strictly chordal graph. Let F^* be such that $H^* = G + F^*$. By construction $F^* \subseteq F$, and the desired properties are verified. ◀

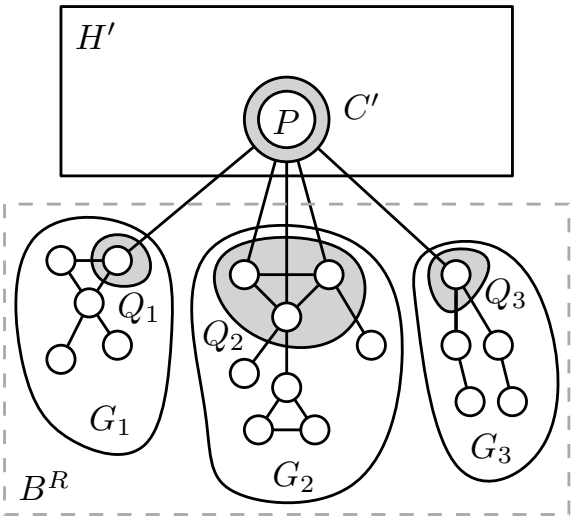


Figure 2 Illustration of the proof of Lemma 12 for $r = 3$. The graph $H^* = (G_1 \cup G_2 \cup G_3, Q_1 \cup Q_2 \cup Q_3) \otimes (H', C')$ is strictly chordal by Lemma 4.

▶ **Rule 3.** *Let $(G = (V, E), k)$ be an instance of Strictly Chordal Completion. If G contains a 1-block-branch B with attachment point P, then remove from G the vertices of $V(B^R)$ and add a clique K of size $min\{|N_B(P)|, k + 1\}$ adjacent to P.*

▶ **Lemma 13** (⋆). *Rule 3 is safe.*

▶ **Lemma 14.** *Let $(G = (V, E), k)$ be a yes-instance of Strictly Chordal Completion, reduced by Rule 2. Let $B_1, \ldots, B_l$ be disjoint 1-block-branches of G with attachment points $P_1, \ldots, P_l$ which have the same neighborhood N in $G \setminus \bigcup_{i=1}^{l} V(B_i)$ and form a disjoint union of cliques $Q_1, \ldots Q_r$ in $G[P_1 \cup \cdots \cup P_l]$. If $\Sigma_{i=1}^{l} |P_i| > 2k+1$, then for every k-completion F of G, N has to be a clique of $H = G+F$. Moreover, if $r > 1$ and $(\Sigma_{i=1}^{l} |P_i|) - max_{1 \leq j \leq r}\{|Q_j|\} > k$ then N is a critical clique of H.*

Proof. If $r = 1$, assume for a contradiction that N is not a clique in H and let $x, y \in N$ such that $xy \notin E(H)$. By our hypothesis, $|Q_1| > 2k + 1$. Recall that since G is reduced by Rule 2, its critical cliques have at most $k + 1$ vertices, in particular each P_i is of size at most $k + 1$. At least one block-branch B_i contains some edge uz with $u \in P_i$ and $z \in V(B_i) \setminus P_i$ (otherwise each block-branch B_i is formed only of P_i, so Q_1 would be a clique module in G, hence contained in some critical clique of G, contradicting the fact that all critical cliques of G are of size at most $k + 1$). In graph G, z is not adjacent to any other block-branch B_j, for any $j \neq i$. Since $\sum_{j \neq i} |P_j| > k$ and F is of size at most k, there must exist some $j \neq i$ and some vertex $v \in P_j$ such that $vz \notin E(H)$. Let us examine the subgraph of H induced by vertices $\{z, u, v, x, y\}$. If z is not adjacent to any of $\{x, y\}$, we obtain a dart. If it is adjacent to exactly one of them, the five vertices induce a gem. Finally, if z is adjacent to both x and y in H, then $\{z, x, v, y\}$ forms a C_4 in this order. In all cases we have a contradiction. It remains that N is a clique in H.

If $r > 1$, suppose for a contradiction that N is not a clique in H, then there exist two vertices $x, y \in N$ such that $xy \notin E(H)$. For any pair of vertices $u_i \in Q_i, u_j \in Q_j, i \neq j$, the set $\{x, u_i, y, u_j\}$ induces a C_4 in G. Thus (u_j, u_i) has to be in F, implying $|F| > k$, a contradiction. Hence N has to be a clique in H. Suppose now that N is not a module in H, then there exists $x, y \in N$ and $z \in V(G) \setminus (N \cup Q_1 \cup \cdots \cup Q_r)$ adjacent to only one of the vertices x or y, say w.l.o.g. x. For any pair of vertices $u_i \in Q_i, u_j \in Q_j, i \neq j$, if $(u_j, u_i) \notin F$

the set $\{x, y, u_i, u_j, z\}$ induces a dart if neither u_i nor u_j is adjacent to z, a gem if one of them is adjacent to z, or if both of them are adjacent to z, $\{y, u_i, u_j, z\}$ induces a C_4 in H. Thus (u_j, u_i) has to be in F, implying $|F| > k$, a contradiction. Hence N must be a module in H. If N is not a critical clique of H, then it is strictly contained in some critical clique N'. A vertex $x \in N' \backslash N$ must have been made adjacent to $N_H[N]$. If x is not in some P_i or $N_{B_i}(P_i)$, then it must have been made adjacent to all vertices of $P_1 \cup \cdots \cup P_l$, implying $|F| > k$, a contradiction. If x is in some P_i or or $N_{B_i}(P_i)$, since $(\Sigma_{i=1}^{l}|P_i|) - max_{1 \leq j \leq r}\{|Q_j|\} > k$, then it must have been made adjacent to the vertices of every $P_j, j \neq i$, implying $|F| > k$, a contradiction. It remains that N is a critical clique in H. ◀

▶ **Rule 4.** *Let $(G = (V, E), k)$ be an instance of* Strictly Chordal Completion *and $B_1, \ldots, B_l$ disjoint 1-block-branches of G with attachment points $P_1, \ldots, P_l$ having the same neighborhood N in $G \backslash \bigcup_{i=1}^{l} V(B_i)$. Assume that $\Sigma_{i=1}^{l}|P_i| > 2k + 1$ and let $Q_1, \ldots Q_r$ be the disjoint union of cliques in $G[P_1 \cup \cdots \cup P_l]$.*

- *If $r = 1$, remove the vertices $\bigcup_{i=1}^{l} V(B_i)$, add two adjacent cliques K and K' of size $k+1$ with neighborhood N and a vertex u_K adjacent to every vertex of K.*
- *If $r > 1$ and $(\Sigma_{i=1}^{l}|P_i|) - max_{1 \leq j \leq r}\{|Q_j|\} > k$, remove the vertices $\bigcup_{i=1}^{l} V(B_i)$ and add two non-adjacent cliques of size $k + 1$ with neighborhood N.*

▶ **Lemma 15** (⋆). *Rule 4 is safe.*

3.3 Reducing the 2-block-branchs

Let B be a 2-block-branch of a graph G reduced by Rule 3, with attachment points P_1 and P_2. We say that B is *clean* if B^R is connected, and that the *length* of a clean 2-block-branch is the length of a shortest path between its two attachment points in $\mathcal{C}(B)$.

▶ **Lemma 16.** *Let $G = (V, E)$ be graph and B a clean 2-block-branch of length at least 3 with attachment points P_1, P_2 that are in different connected components of $G \backslash V(B^R)$. There exists an optimal completion F of G such that:*

- *The set of vertices of B affected by F is included in $P_1 \cup N_B(P_1) \cup P_2 \cup N_B(P_2)$,*
- *In $H = G + F$, the vertices of $N_B(P_1)$ (resp. $N_B(P_2)$) are all adjacent to the same vertices of $V(G) \backslash V(B^R)$.*

Proof. Let F be an optimal completion of G, and $H = G + F$. Recall that by hypothesis P_1 and P_2 are in different connected components of $G \setminus V(B^R)$ and let G_1 and G_2 be these components. Consider the graphs $H_1 = H[V(G_1) \cup V(B^R)]$, $H_2 = H[V(G_2) \cup V(B^R)]$ and $H_3 = H[V(G) \backslash (V(G_1) \cup V(G_2) \cup V(B^R))]$, which are strictly chordal graphs by heredity. Let C_i be the critical clique of H_i which contains P_i, and $C_i' = C_i \backslash V(B^R)$, $i \in \{1, 2\}$. By Lemma 11, C_i' is a critical clique of $H_i' = H_i \backslash V(B^R)$ or is in exactly one maximal clique of H_i'. Since P_1 are P_2 are at distance at least 3 to each other, $N_B(P_1)$ and $N_B(P_2)$ are disjoint. Since B^R is connected, Lemma 10 gives that $N_B(P_1)$ and $N_B(P_2)$ are cliques in G and are critical cliques of B^R or are each in exactly one maximal clique of B^R. By Lemma 4, the graph H^* corresponding to the disjoint union of $((B^R, N_B(P_1)) \otimes (H_1', C_1')), N_B(P_2)) \otimes (H_2', C_2')$ and H_3 is strictly chordal and the completion $F^* \subseteq F$, such that $H^* = G + F^*$, respects the desired properties. ◀

▶ **Rule 5.** *Let $(G = (V, E), k)$ be an instance of* Strictly Chordal Completion *and B a clean 2-block-branch of G of length at least 3 with attachment points P_1, P_2 that are in different connected components of $G \backslash V(B^R)$. Remove the vertices of B^R, and add two new cliques K_1 and K_2 of size respectively $min\{|N_B(P_1)|, k + 1\}$ and $min\{|N_B(P_2)|, k + 1\}$ with the edges $(P_1 \times K_1), (K_1 \times K_2), (K_2 \times P_2)$.*

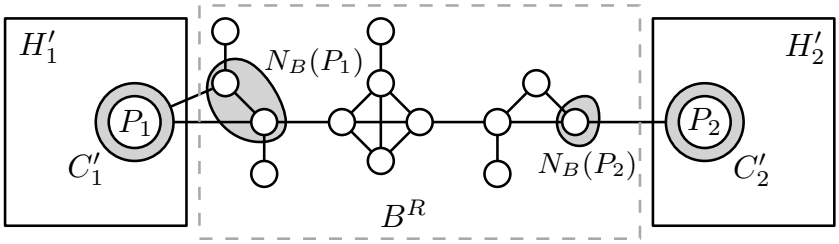


Figure 3 Illustration of the proof of Lemma 16. The graph $H^* = ((B^R, N_B(P_1)) \otimes (H'_1, C'_1)), N_B(P_2)) \otimes (H'_2, C'_2)$ is strictly chordal by Lemma 4.

▶ **Lemma 17** (⋆). *Rule 5 is safe.*

4 Size of the kernel

We first state that reduction rules involving block-branches can be applied in polynomial time.

▶ **Lemma 18.** *Given an instance $(G = (V, E), k)$ of* Strictly Chordal Completion*, Rules 3 to 5 can be applied in polynomial time.*

Proof. We rely on a linear time computation of the critical clique graph $\mathcal{C}(G)$ of G [29] and a linear time recognition algorithm for block graphs [1]. We show that we can enumerate all 1-block-branches and 2-block-branches in polynomial time. Since an attachment point is by definition a critical clique of G, one can detect 1-block-branches by removing a critical clique of $\mathcal{C}(G)$ and look among the remaining connected components those that induce a connected block graph together with P (in $\mathcal{C}(G)$). Considering a maximal set of such components together with P gives a 1-block-branch B. We proceed similarly to detect clean 2-block-branches by removing a pair of critical cliques P_1, P_2 of $\mathcal{C}(G)$, and look among the remaining connected components those that induce a connected block graph (in $\mathcal{C}(G)$) together with P_1 and P_2. Such components together with P_1 and P_2 gives a clean 2-block-branch B. Recall that Rule 5 only applies if P_1 and P_2 are not in the same component of $G \backslash V(B^R)$, which can easily be verified. Since there is $O(|V(G)|)$ critical cliques, the result follows. ◀

▶ **Lemma 19.** *Let $G = (V, E)$ be a connected block graph, a set $A \subseteq V(G)$ and T_A be a minimal connected induced subgraph of G that spans all vertices of A. Denote by $f(T_A)$ the set of vertices of degree at least 3 in T_A. We have:*

- **(i)** *The subgraph T_A is unique,*
- **(ii)** $|f(T_A) \setminus A| \leq 3 \cdot |A|$,
- **(iii)** *The graph $T_A \backslash (A \cup f(T_A))$ contains at most $2 \cdot |A|$ connected components.*

Proof. A convenient way to represent the tree-like structure of block graph G is its block-cut tree $T_{BC}(G)$. Recall that the block-cut tree of a graph G has two types of nodes: the *block nodes* correspond to blocks of G (i.e, bi-connected components which, in our case, are precisely the maximal cliques of G) and the *cut nodes* correspond to cut-vertices of G. We put an edge between a cut node and a block node in $T_{BC}(G)$ if the corresponding cut-vertex belongs to the corresponding block of G.

Each vertex of G that is not a cut-vertex belongs to a unique block of G. Therefore we can map each vertex v of $V(G)$ on a unique node $n(v)$ of $T_{BC}(G)$ as follows: if v is a cut-vertex, we map it on its corresponding cut node in $T_{BC}(G)$, otherwise we map it on the

block node corresponding to the unique block of G containing v. Observe that, for any two vertices $u, v \in A$, any (elementary) path P from u to v in G corresponds to the path P_{BC} from $n(u)$ to $n(v)$ in $T_{BC}(G)$. In particular, P contains all vertices corresponding to cut nodes of P_{BC}, therefore T_A must contain all these vertices. Altogether, the vertices of T_A are precisely the vertices of A, plus all cut-vertices of G corresponding to the cut nodes of such paths, implying the unicity of T_A.

Let $T_{BC}(A)$ denote the subtree of $T_{BC}(G)$ spanning all nodes of $n(A) = \{n(a) \mid a \in A\}$. We count the vertices of $f(T_A) \setminus A$, so let a' be such a vertex. By the previous observation, it is a cut-vertex of G, so $n(a')$ is a cut node of $T_{BC}(A)$. Let v be a neighbor of a' in T_A. By construction of T_A and $T_{BC}(A)$, there is a block node b adjacent to $n(a')$ in $T_{BC}(A)$, such that v and a' are in the maximal clique of G corresponding to node b. Moreover, v is in A, or v is a cut-vertex of G such that the cut node $n(v)$ is adjacent to b in $T_{BC}(A)$. Hence we have:

1. $n(a')$ is of degree at least 3 in $T_{BC}(A)$, or
2. $n(a')$ is the neighbor of a block node b of degree at least three in $T_{BC}(A)$, or, if none of these hold, then
3. $n(a')$ has exactly two neighbors b and b' in $T_{BC}(A)$, the corresponding maximal cliques of G contain at least p vertices of A, where p plus the degrees of b and b' in T_{BC} is at least three.

Let k be the number of leaves of $T_{BC}(A)$. Observe that for any leaf l of $T_{BC}(A)$, there is some $a \in A$ such that $n(a) = l$. Choose for each leaf l a unique vertex $a_l \in A$ such that $n(a_l) = l$, we call a_l a leaf vertex. Note that the number of vertices $a' \in f(A)$ corresponding to the first two items of the above enumeration is upper bounded by $3k$. Indeed, $n(a')$ is incident to an edge of $T_{BC}(A)$, having an end node of degree at least 3. One can easily check that, in any tree of k leaves, the number of such edges is bounded by $3k$ (this can be shown by induction on the number of leaves of the tree, adding a new leaf node at a time). We count now the vertices $a' \in f(A)$ of the third type. By the third item, a' has at least one neighbor $a \in A$ in graph T_A, such that a is *not* a leaf vertex. Observe that a can be in the neighborhood of at most two vertices $a' \in f(A)$ of this third type. Altogether it follows that $|f(A) \setminus A \leq 3k + 2(|A| - k) \leq 3 \cdot |A|$.

To prove the third item, the number of components of $T_A \setminus (A \cup f(A))$, we visualize again the situation in $T_{BC}(A)$. Recall that for any node of degree at least three in $T_{BC}(A)$ it is either a cut node, in which case it is in $f(A)$ (the first case of the enumeration above), or it is a block node but then all its neighbors in $T_{BC}(A)$ correspond to vertices of $f(A)$ (the second case of the enumeration above). Therefore the components of $T_A \setminus f(A)$ correspond to the components of $T_{BC}(A)$ after removal of all nodes of degree at least 3. Hence the number of such components is upper bounded by $2k$. Removing the leaf vertices of A does not increase the number of components, and the removal of each other vertex of A increases the number of components by at most one. Thus $T_A \setminus (A \cup f(A))$ has at most 2 $cdot|A|$ components, concluding the proof of our lemma. ◀

▶ **Lemma 20** ([29]). *Let $G = (V, E)$ be a graph and $H = G \triangle \{uv\}$ for $u, v \in V$, then $\mathcal{K}(H) \leq \mathcal{K}(G) + 2$.*

▶ **Theorem 21.** *Strictly Chordal Completion admits a kernel with $O(k^3)$ vertices.*

Proof. Let $(G = (V, E), k)$ be a reduced yes-instance of Strictly Chordal Completion, F a k-completion of G and $H = G + F$. We assume that G is connected, the following arguments can be easily adapted if this is not the case by summing over all connected components of G. The graph H is strictly chordal, thus the graph $\mathcal{C}(H)$ is a block graph.

We first show that $\mathcal{C}(H)$ has $O(k^2)$ vertices. We say that a critical clique of $\mathcal{C}(H)$ is affected if it contains a vertex affected by F. Let A be the set of affected critical cliques of $\mathcal{C}(H)$. Since $|F| \leq k$, we have $|A| \leq 2k$. Let T be the connected minimal subgraph of H that spans all critical cliques of A, and A' the set of vertices of degree at least 3 in T.

We first show that $|V(T)|$ is $O(k)$. By Lemma 19, $|A' \setminus A| \leq 3 \cdot |A| \leq 6k$. The connected components of the graph $T \backslash (A \cup A')$ are paths since every vertex is of degree at most 2 and by Lemma 19 there are at most $4k$ such paths. Let R be one of these paths, it is composed of unaffected critical cliques, thus there exists a 2-block-branch B of G that contains R. Moreover, the extremities of R are the attachment points of B, which have been reduced by Rule 5. Thus R is of length at most 3. It remains that $|V(T)| = 2k + 6k + 4 \cdot 4k = O(k)$.

We will now show that $\mathcal{C}(H) \backslash V(T)$ contains $O(k^2)$ critical cliques. First observe that each connected component of $\mathcal{C}(H) \backslash V(T)$ is adjacent to some vertices of T since the graph is reduced by Rule 1. Since $\mathcal{C}(H)$ is a block graph, connected components of $\mathcal{C}(H) \backslash V(T)$ are adjacent to either a critical clique of T or a maximal clique of T (else, there would be a diamond in $\mathcal{C}(H)$). We claim that there are $O(k^2)$ critical cliques in the connected components of $\mathcal{C}(H) \backslash V(T)$ adjacent to maximal cliques of T. Since T is a block graph and $|V(T)| = O(k)$, there are $O(k)$ maximal cliques in T. Moreover, there is only one connected component of $\mathcal{C}(H) \backslash V(T)$ adjacent to each maximal clique, otherwise there would be a diamond in $\mathcal{C}(H)$. Take K a maximal clique of T and let X be its adjacent connected component of $\mathcal{C}(H) \backslash V(T)$. Observe that X has to be an union of 1-block-branches and their attachment points form a clique. By Rule 4, there are at most $2k + 1$ 1-block-branches in X and each one has been reduced by Rule 3, thus X contains at most $4k + 2$ critical cliques.

Finally, we claim that there are $O(k^2)$ critical cliques in the connected components of $\mathcal{C}(H) \backslash V(T)$ adjacent to critical cliques of T. First take a critical clique of $T \backslash A$ and its adjacent connected components of $\mathcal{C}(H) \backslash V(T)$. Observe that they form a 1-block-branch, reduced by Rule 3, thus the adjacent connected component consists in only one critical clique. Next, take a critical clique a of A, and let $C_1, \ldots, C_r$ be the connected components of $\mathcal{C}(H) \backslash V(T)$ adjacent to a and Y the union of these connected components. Observe that each C_i has to be an union of 1-block-branches and their attachment points form a clique Q_i. Let $B_1, \ldots, B_l$ with attachment points $P_1, \ldots, P_l$ be the 1-block-branches of Y. By Rule 4, there are at most $2k + 1$ 1-block-branches by connected component and if $\Sigma_{i=1}^{l} |P_i| > 2k + 1$, then $(\Sigma_{i=1}^{l} |P_i|) - max_{1 \leq j \leq r}\{|Q_j|\} \leq k$, implying that there are at most $3k + 1$ 1-block-branches in C. Each of these 1-block-branches is reduced by Rule 3, hence Y contains at most $6k + 2$ critical cliques in total.

We conclude that $|V(\mathcal{C}(H))| = O(k^2)$ and by Lemma 20, we have $|V(\mathcal{C}(G))| \leq |V(\mathcal{C}(H))| + 2k$, therefore by Rule 2, $|V(G)| = O(k^3)$. To conclude, we claim that a reduced instance can be computed in polynomial time. Indeed, Lemma 8 states that it is possible to reduce exhaustively a graph under Rules 1 and 2. Once this is done, it remains to apply exhaustively Rules 3 to 5 which is ensured by Lemma 18. ◀

5 Kernels for edition and deletion

In this section we present the kernel for Strictly Chordal Editing, all the following arguments also hold for Strictly Chordal Deletion. It is clear that Rules 1 and 2 are also safe for Strictly Chordal Editing, as well as Lemma 11. Lemma 12 also holds although the proof needs to be adapted to take in consideration the possibility of disconnecting the attachment point of a 1-block-branch B and B^R. It follows that Rule 3 is safe for the edition version. Next, Lemma 14 holds for edition with similar arguments for the

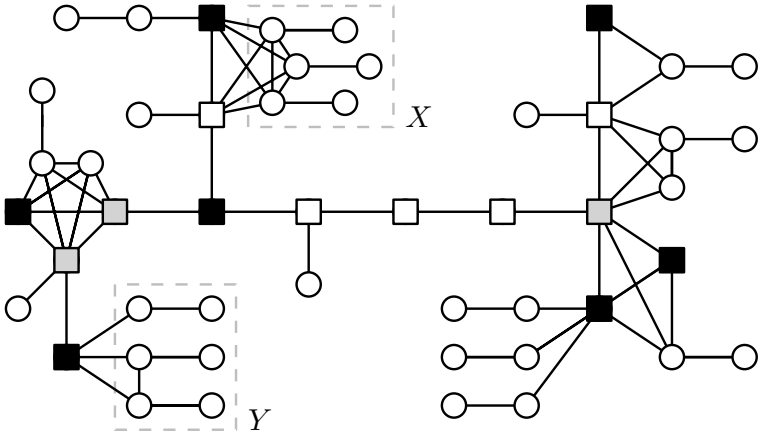


Figure 4 Illustration of the critical clique graph of an reduced instance in the proof of Theorem 21. Square nodes correspond to vertices of T, the ones filled in black are vertices of A, the ones filled in grey are vertices of $A' \backslash A$.

proof, implying the safeness of Rule 4 for edition. The main difference lies in Rule 5 handling 2-block-branches. Indeed, in this case, an optimal edition may affect vertices that are not contained in an attachment point nor their neighborhood in the branch. However, this case can be dealt with with more intricate arguments (Rule 6).

Finally, we can observe that for a clean 2-block-branch B of some graph G, an edition can remove edges and disconnect B. We thus have to consider 2-block-branches whose attachment points lie in the same connected component of $G \backslash V(B^R)$. We hence have to take this into consideration for our new reduction rule. To that aim, we use a minimum-sized (P_1, P_2)-cut of B where P_1 and P_2 are its attachment points. More precisely, we define *min-cut*(B) as a set $M \subseteq E(B)$ of minimum size such that P_1 and P_2 are not in the same connected component of $B - M$. We can observe that *min-cut*(B) contains the edges between a pair of consecutive critical cliques of a shortest path between P_1 and P_2 and the edges between one of these critical cliques and the critical cliques they have in their common neighborhood.

▶ **Observation 22.** *Let F be an optimal edition of G and $F_1 \subseteq F$. If F_2 is an optimal edition of the graph $G \triangle F_1$, then $F_1 \cup F_2$ is an optimal edition of G.*

▶ **Lemma 23.** *Let $G = (V, E)$ be graph and B a clean 2-block-branch of length at least $k + 4$ with attachment points P_1, P_2. There exists an optimal edition F of G such that:*

- *If P_1 and P_2 are not in the same connected component of $B \triangle F$, then F may contain edges of min-cut(B).*
- *In each case, the other vertices of B affected by F are included in $P_1 \cup N_B(P_1) \cup P_2 \cup N_B(P_2)$,*
- *In $G \triangle F$ the vertices of $N_B(P_1)$ (resp. $N_B(P_2)$) are all adjacent to the same vertices of $V(G) \backslash V(B^R)$.*

Proof. Let F be an optimal edition of G, $H = G \triangle F$. Let C_1 (resp. C_2) be the critical clique of H which contains P_1 (resp. P_2), $C_1' = C_1 \backslash (B^R)$ and $C_2' = C_2 \backslash (B^R)$. We first consider the case where B is not connected in $B \triangle F$. If $F_1 = (P_1 \times N_B(P_1)) \subseteq F$, consider the graph $G_1 = G \triangle F_1$. Observe that $B_1 = B \backslash P_1$ is a 1-block-branch of G_1 with attachment point P_2. By Lemma 12 there exists an optimal edition F_2 of G_1 where the vertices of B_1 affected by F_2 are in $P_2 \cup N_{B_1}(P_2)$. By Observation 22, $F_1 \cup F_2$ is an optimal edition and it satisfies the desired properties. The same arguments can be used if $(P_2 \times N_B(P_2)) \subseteq F$.

If F contains neither $(P_1 \times N_B(P_1))$ nor $(P_2 \times N_B(P_2))$, since F is optimal, it must contain $F_1 = \textit{min-cut}(B)$. We consider the graph $G_1 = G \triangle F_1$, there are two components in $G_1[V(B)]$, say B_1 the one containing P_1 and B_2 the other one containing P_2. These connected components are 1-block-branches with attachment points P_1 and P_2. As before Lemma 12 applies on the 1-block-branches B_1 and B_2, thus there is an optimal edition F_2 of G_1 where the only vertices of B_1 and B_2 affected are $(P_1 \times N_B(P_1))$ and $(P_2 \times N_B(P_2))$. Thus, the edition $F_1 \cup F_2$ is an optimal edition of G that respects the desired properties.

In the case where B is connected in $B \triangle F$, observe that C'_1 and C'_2 have to be in different connected components of $H \backslash V(B^R)$. Otherwise there would be a shortest path p_H from $c_1 \in C'_1$ to $c_2 \in C'_2$ in $H \backslash V(B^R)$ (of length potentially 0 if $c_1 = c_2$) and a shortest path p_B from c_1 to c_2 in B of length at least $k+4$. There is no edge between the two paths in G, so $H[V(p_B) \cup V(p_H)]$ admits at least one cycle of length at least 4, contradiction. Since B^R is connected, by Lemma 10, $N_B(P_1)$ and $N_B(P_2)$ are cliques in G, $N_B(P_1)$ and $N_B(P_2)$ are critical cliques of B^R or each are in exactly one maximal clique of B^R. By Lemma 11, C'_1 and C'_2 are either critical cliques of $H' = H \backslash V(B^R)$ or are in exactly one maximal clique of the connected component H'_1 (resp. H'_2) of H' which contains C'_1 (resp. C'_2). Let H_3 be the union of connected components of H that do not contain B. By Lemma 4, the graph H^* corresponding to the disjoint union of $((B^R, N_B(P_1)) \otimes (H'_1, C'_1)), N_B(P_2)) \otimes (H'_2, C'_2)$ and H_3 is strictly chordal and the completion F^* such that $H^* = G + F^*$ respects the desired properties. ◀

▶ **Rule 6.** *Let $(G = (V, E), k)$ be an instance of* Strictly Chordal Editing *and B a clean 2-block-branch of G of length at least $k+4$ with attachment points P_1, P_2. Then remove the vertices of B^R, and add the following path of $k+5$ cliques:*

$$K_{min\{|N_B(P_1)|,k+1\}} - K_{k+1} - K_1 - K_{|\textit{min-cut}(B)|} - K^1_{k+1} - \cdots - K^k_{k+1} - K_{min\{|N_B(P_2)|,k+1\}}$$

where K_n is the clique of size n and $K_{min\{|N_B(P_1)|,k+1\}}$ (resp. $K_{min\{|N_B(P_2)|,k+1\}}$) is adjacent to P_1 (resp. P_2).

▶ **Lemma 24** ($\star$). *Rule 6 is safe.*

Notice that Lemma 18 allows to detect any clean 2-block-branch. For the size of the kernel, the proof is similar, however, in this case the paths are of length $O(k)$, thus there are $O(k^2)$ vertices and maximal cliques in the inclusion-minimal subgraph spanning the affected critical cliques. Thus a reduced graph contains $O(k^4)$ vertices.

▶ **Theorem 25.** *Strictly Chordal Editing and Strictly Chordal Deletion admits a kernel with $O(k^4)$ vertices.*

6 Conclusion

We presented polynomial size kernels for the three variants of strictly chordal edge modification problems. Our conviction is that the approach based on decompositions of the target class, combined with the ability of reducing the size of the bags of the decomposition and of limiting the number of affected bags to $O(k)$ is a promising starting point for edge modification problems, especially into subclasses of chordal graphs. The technique has been employed especially for classes that admit a tree-like decompositions with disjoint bags (e.g., 3-leaf power and trivially perfect graphs [3, 17]), also for other types of tree-like decompositions with non-disjoint bags (e.g., ptolemaic graphs [12]). We generalize it here to scrictly chordal graphs, that have a decomposition into disjoint bags as nodes of a block graph.

The difficulty is that, at this stage, each class requires ad-hoc arguments and reduction rules, based on its specific decomposition. An ambitious goal would be to obtain a generic algorithm for edge modification problems into any class of chordal graphs, plus a finite set of obstructions, as conjectured by Bessy and Perez [4]. We also ask whether 4-leaf power completion, deletion and editing problems admit a polynomial kernel.

References

1 Hans-Jürgen Bandelt and Henry Martyn Mulder. Distance-hereditary graphs. *J. Comb. Theory, Ser. B*, 41(2):182–208, 1986. doi:10.1016/0095-8956(86)90043-2.

2 Gabriel Bathie, Nicolas Bousquet, and Théo Pierron. (sub)linear kernels for edge modification problems towards structured graph classes. *CoRR*, abs/2105.09566, 2021. arXiv:2105.09566.

3 Stéphane Bessy, Christophe Paul, and Anthony Perez. Polynomial kernels for 3-leaf power graph modification problems. *Discret. Appl. Math.*, 158(16):1732–1744, 2010. doi:10.1016/j.dam.2010.07.002.

4 Stéphane Bessy and Anthony Perez. Polynomial kernels for proper interval completion and related problems. *Inf. Comput.*, 231:89–108, 2013. doi:10.1016/j.ic.2013.08.006.

5 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009. doi:10.1016/j.jcss.2009.04.001.

6 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. doi:10.1137/120880240.

7 Hans L. Bodlaender, Stéphan Thomassé, and Anders Yeo. Kernel bounds for disjoint cycles and disjoint paths. *Theor. Comput. Sci.*, 412(35):4570–4578, 2011. doi:10.1016/j.tcs.2011.04.039.

8 Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Inf. Process. Lett.*, 58(4):171–176, 1996. doi:10.1016/0020-0190(96)00050-6.

9 Leizhen Cai and Yufei Cai. Incompressibility of h-free edge modification problems. *Algorithmica*, 71(3):731–757, 2015. doi:10.1007/s00453-014-9937-x.

10 Yixin Cao and Dániel Marx. Chordal editing is fixed-parameter tractable. *Algorithmica*, 75(1):118–137, 2016. doi:10.1007/s00453-015-0014-x.

11 Christophe Crespelle, Pål Grønås Drange, Fedor V. Fomin, and Petr A. Golovach. A survey of parameterized algorithms and the complexity of edge modification. *CoRR*, abs/2001.06867, 2020. arXiv:2001.06867.

12 Christophe Crespelle, Benjamin Gras, and Anthony Perez. Completion to chordal distance-hereditary graphs: a quartic vertex-kernel. Accepted for publication at International Workshop on Graph-Theoretic Concepts in Computer Science, 2021.

13 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5. Springer, 2015.

14 Michael Dom, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Error compensation in leaf power problems. *Algorithmica*, 44(4):363–381, 2006. doi:10.1007/s00453-005-1180-z.

15 Michael Dom, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Closest 4-leaf power is fixed-parameter tractable. *Discret. Appl. Math.*, 156(18):3345–3361, 2008. doi:10.1016/j.dam.2008.01.007.

16 Pål Grønås Drange and Michał Pilipczuk. A polynomial kernel for trivially perfect editing. *Algorithmica*, 80(12):3481–3524, 2018. doi:10.1007/s00453-017-0401-6.

17 Maël Dumas, Anthony Perez, and Ioan Todinca. A cubic vertex-kernel for trivially perfect editing. *CoRR*, abs/2105.08549, 2021. arXiv:2105.08549.

18 Martin Charles Golumbic and Uri N. Peled. Block duplicate graphs and a hierarchy of chordal graphs. *Discret. Appl. Math.*, 124(1-3):67–71, 2002. doi:10.1016/S0166-218X(01)00330-4.

19 Sylvain Guillemot, Frédéric Havet, Christophe Paul, and Anthony Perez. On the (non-)existence of polynomial kernels for P_l-free edge modification problems. *Algorithmica*, 65(4):900–926, 2013. doi:10.1007/s00453-012-9619-5.

20 Jiong Guo. Problem kernels for np-complete edge deletion problems: Split and related graphs. In Takeshi Tokuyama, editor, *Algorithms and Computation, 18th International Symposium, ISAAC 2007, Sendai, Japan, December 17-19, 2007, Proceedings*, volume 4835 of *Lecture Notes in Computer Science*, pages 915–926. Springer, 2007. doi:10.1007/978-3-540-77120-3_79.

21 Haim Kaplan, Ron Shamir, and Robert Endre Tarjan. Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs. *SIAM J. Comput.*, 28(5):1906–1922, 1999. doi:10.1137/S0097539796303044.

22 William Kennedy. Strictly chordal graphs and phylogenetic roots. Master's thesis, University of Alberta, 2005.

23 William Kennedy, Guohui Lin, and Guiying Yan. Strictly chordal graphs are leaf powers. *Journal of Discrete Algorithms*, 4(4):511–525, 2006. doi:10.1016/j.jda.2005.06.005.

24 Stefan Kratsch and Magnus Wahlström. Two edge modification problems without polynomial kernels. *Discret. Optim.*, 10(3):193–199, 2013. doi:10.1016/j.disopt.2013.02.001.

25 Mirko Křivánek and Jaroslav Morávek. Np-hard problems in hierarchical-tree clustering. *Acta Informatica*, 23(3):311–323, 1986. doi:10.1007/BF00289116.

26 Lilian Markenzon and Christina Fraga Esteves Maciel Waga. New results on ptolemaic graphs. *Discret. Appl. Math.*, 196:135–140, 2015. doi:10.1016/j.dam.2014.03.024.

27 Assaf Natanzon. *Complexity and approximation of some graph modification problems*. University of Tel-Aviv, 1999.

28 Naomi Nishimura, Prabhakar Ragde, and Dimitrios M. Thilikos. On graph powers for leaf-labeled trees. *J. Algorithms*, 42(1):69–108, 2002. doi:10.1006/jagm.2001.1195.

29 Fábio Protti, Maise Dantas da Silva, and Jayme Luiz Szwarcfiter. Applying modular decomposition to parameterized cluster editing problems. *Theory Comput. Syst.*, 44(1):91–104, 2009. doi:10.1007/s00224-007-9032-7.

30 Ron Shamir, Roded Sharan, and Dekel Tsur. Cluster graph modification problems. *Discret. Appl. Math.*, 144(1-2):173–182, 2004. doi:10.1016/j.dam.2004.01.007.

On Extended Formulations For Parameterized Steiner Trees

Andreas Emil Feldmann
Department of Applied Mathematics, Charles University, Prague, Czech Republic

Ashutosh Rai
Department of Mathematics, IIT Delhi, India

Abstract

We present a novel linear program (LP) for the Steiner Tree problem, where a set of terminal vertices needs to be connected by a minimum weight tree in a graph $G = (V, E)$ with non-negative edge weights. This well-studied problem is NP-hard and therefore does not have a compact extended formulation (describing the convex hull of all Steiner trees) of polynomial size, unless P=NP. On the other hand, Steiner Tree is fixed-parameter tractable (FPT) when parameterized by the number k of terminals, and can be solved in $O(3^k|V| + 2^k|V|^2)$ time via the Dreyfus-Wagner algorithm. A natural question thus is whether the Steiner Tree problem admits an extended formulation of comparable size.

We first answer this in the negative by proving a lower bound on the extension complexity of the Steiner Tree polytope, which, for some constant $c > 0$, implies that no extended formulation of size $f(k)2^{cn}$ exists for any function f. However, we are able to circumvent this lower bound due to the fact that the edge weights are non-negative: we prove that Steiner Tree admits an integral LP with $O(3^k|E|)$ variables and constraints. The size of our LP matches the runtime of the Dreyfus-Wagner algorithm, and our poof gives a polyhedral perspective on this classic algorithm. Our proof is simple, and additionally improves on a previous result by Siebert et al. [2018], who gave an integral LP of size $O((2k/e)^k)|V|^{O(1)}$.

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

Keywords and phrases Steiner trees, integral linear program, extension complexity, fixed-parameter tractability

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.18

Funding *Andreas Emil Feldmann*: Supported by the Czech Science Foundation GAČR (grant #19-27871X), and by the Center for Foundations of Modern Computer Science (Charles Univ. project UNCE/SCI/004).
Ashutosh Rai: Major part of the work was done when the author was at Charles University, funded by Center for Foundations of Modern Computer Science (Charles Univ. project UNCE/SCI/004).

Acknowledgements We would like to thank Petr Kolman and Hans Raj Tiwary for helpful discussions.

1 Introduction

A central topic in combinatorial optimization is to determine whether a polynomially solvable optimization problem admits a *compact extended formulation*, i.e., if it admits a polytope of polynomial size describing the convex hull of all feasible solutions. The existence of such a polytope means that the corresponding problem can be solved efficiently using linear programming (LP) solvers. This has been a very fruitful research direction from its beginnings all the way to the present day, of which we give a brief overview below (see the surveys [10, 18]) to set the stage for our contribution on the NP-hard Steiner Tree problem.

If $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ is an n-dimensional polytope then a d-dimensional polytope $Q \subseteq \mathbb{R}^d$ is an *extension* of P if P is the projection of Q onto the variables of P, i.e., $P = \{x \mid \exists y : (x, y) \in Q\}$. The *size* of a polytope is the number of its facets (i.e., the number

of inequalities in its description), and its *extension complexity* is the minimum size of any of its extensions. An *extended formulation* of an optimization problem is an extension of the polytope given by the convex hull of all characteristic vectors of feasible solutions to the problem. An extended formulation is said to be *compact* if its size is polynomially bounded in the input size. In particular, once a compact extended formulation has been found, we may optimize over it using an LP solver and then project the LP solution to a solution of the problem. Given a modern LP solver this can be used to obtain very efficient algorithms to solve the problem.

The well known Weyl-Minkowski Theorem [28] states that every convex hull of a finite set of vectors (of solution vectors in particular) can be described as a polytope, i.e., every optimization problem with a finite number of solutions admits an extended formulation, which however may not be compact. There are problems for which compact extended formulations exist, for example spanning trees [23] and perfect matchings in planar graphs [3]. On the other hand, there are some polynomially solvable problems that do not admit compact extended formulations, notably for matchings in general graphs: Rothvoß [26] proved that the matching polytope has exponential extension complexity.

For NP-hard problems we do not expect to find compact extended formulations, since this would imply P=NP given that LPs can be solved in polynomial time. However, in the past few decades the development of the theory of fixed-parameter tractability (cf. [11]) has given us a more fine-grained view, where we differentiate between NP-hard problems that are more tractable than others: it can be shown that for some NP-hard problems the expected super-polynomial runtime overhead can be restricted to a *parameter* $k \in \mathbb{N}$, which describes some property of the input, while the runtime remains polynomial in the input size n. Formally, a problem is called *fixed-parameter tractable (FPT)* if it can be solved by an algorithm with runtime $f(k)n^{O(1)}$ for any input of size n and parameter k, where $f : \mathbb{N} \to \mathbb{N}$ is some computable function. Analogous to the complexity class P, which captures all problems that are polynomially solvable, the class FPT contains all problems that are FPT, and can thus be considered more tractable than other NP-hard problems for which FPT algorithms do not exist.

In light of the research on compact extended formulation for problems in P, a very natural question becomes: what problems in FPT admit extended formulations of size $f(k)n^{O(1)}$? Or even more ambitious: suppose that a problem in FPT can be solved in time $O(g(k)n^c)$ for some specific function g and constant c (say $g(k) = 2^k$ and $c = 1$). Is there an extended formulation for which the size matches the running time $O(g(k)n^c)$ of the algorithm? This question has for instance been studied by Buchanan [4] for the Vertex Cover problem. He gave an extended formulation of size $O(1.47^k + kn)$ where k is the solution size, which does not yet match the currently best running time of $O(1.2738^k + kn)$ for the problem [8]. For the parameterization by the treewidth t of the input graph, an extended formulation of size $O(2^t n)$ exists [5], which in this case also matches the fastest algorithm for this parameter [11]. A generalization of this results is by Kolman et al. [20], who give extended formulations for CSPs parameterized by the treewidth.

Given a graph $G = (V, E)$ and a set of *terminals* $T \subseteq V$, a *Steiner tree* of G is an inclusion-wise minimal connected subgraph of G containing all the terminals (it has to be a tree where all the leaves are terminals). In this paper, we consider the Steiner Tree problem, where we are given an undirected graph G with non-negative edge weights $w : E \to \mathbb{R}^+$ and a set of terminals $T \subseteq V$, and the aim is to find a minimum weight Steiner tree for the terminal set T in G. This problem is known to be NP-hard [14], and has a lot of applications including VLSI routing [9, 17], phylogenetic tree construction [16], and network

routing [22]. A well-studied parameter for this problem is the number of terminals $k = |T|$, for which the classic Dreyfus-Wagner algorithm [12] solves the problem in $O(3^k n + 2^k n^2)$ time. In light of the above discussion, we wish to compare this runtime to the extension complexity of the Steiner Tree polytope, defined as

$$ST(G,T) := \text{conv.hull}\ \{x^{E(H)} \in \{0,1\}^{|E(G)|} \mid H \text{ is a Steiner tree for } T \text{ in } G\},$$

where for a subgraph H of G, $x^{E(H)}$ represents the characteristic vector of length $|E(G)|$ for the set $E(H)$, and conv.hull(S) denotes the convex hull of a set S of vectors.

Our first result is a lower bound showing that the extension complexity of the Steiner Tree polytope is exponential, and thus, in contrast to the Vertex Cover problem, in general we cannot hope to find an extended formulation of $f(k)n^{O(1)}$ size. In fact this is true even if the number k of terminals is constant.

▶ **Theorem 1.** *There exists a constant $c > 0$ such that for any function f, there exists a graph G on n vertices such that the extension complexity of the Steiner Tree polytope $ST(G,T)$ is at least $f(k)2^{cn}$, where $k = |T|$.*[1] *In particular, the extension complexity of $ST(G,T)$ is $2^{\Omega(n)}$ for some graph G with n vertices even when $|T| = 2$.*

An extended formulation asymptotically matching this exponential lower bound can be obtained for Steiner Tree via the matroid polytope [21]. However, in contrast to Theorem 1 we give an *integral* LP for Steiner Tree, i.e., an LP for which each extreme point is a $(0,1)$-vector: our main result is that for any graph G and terminal set T, there is an LP of $f(k)n^{O(1)}$ size that optimizes over a polyhedron, which contains an extension of $ST(G,T)$, and every optimum solution to the LP given by *non-negative* edge weights projects to a point of $ST(G,T)$. In other words, we find a polyhedron that describes the lower envelope of the Steiner Tree polytope by projecting to all the characteristic vectors of optimum Steiner trees for non-negative edge weights. Thus, it suffices to solve this LP to solve the Steiner Tree problem given its definition, and so we are able to circumvent the lower bound of Theorem 1.

▶ **Theorem 2.** *For any Steiner Tree instance on a graph $G = (V,E)$ with n vertices, m edges, and k terminals $T \subseteq V$, there is a $3^k m$-dimensional polyhedron Q given by $2^k n + 3^k m$ constraints, such that $ST(G,T) \subseteq \{x \in \mathbb{R}^{|E|} \mid \exists y : (x,y) \in Q\}$ and optimizing over Q with any non-negative edge weight function $w : E \to \mathbb{R}^+$ gives a point of Q that projects to a point of $ST(G,T)$.*

Siebert et al. [27] showed that Steiner Tree can be solved using $O((2k/e)^k)$ polynomial-sized integral LPs. Each solution vector also projects to a Steiner tree. It is possible to write all these LPs into one integral LP (by a result of Balas [1, 2], see Theorem 10), which then has size $(2k/e)^k n^{O(1)} = 2^{O(k \log k)} n^{O(1)}$. Theorem 2 improves on this result by making the size single exponential. Moreover, the size of our LP matches the runtime of the Dreyfus-Wagner algorithm, and thus Theorem 2 provides an alternative way of solving the Steiner Tree problem in comparable runtime via linear programming. In another work, Martin et al. [24] show how to obtain an LP having size linear in the running time of any dynamic programming algorithm. Even though their result can be used to solve the Steiner Tree problem due to the Dreyfus-Wagner dynamic program [12], their LP is not an extension of the Steiner

[1] We remark that compared to parameterized runtime lower bounds, this lower bound is unconditional and therefore the function f is not restricted to be computable.

Tree polytope, i.e., it does not project down to $ST(G,T)$. Instead their LP describes the states of the dynamic program and thus gives an indirect way to encode a solution via an LP. On the other hand, our LP solutions describe the vertices of $ST(G,T)$ via a direct projection.

1.1 Our techniques

On a high level our LP of Theorem 2 takes its inspiration from flow based formulations. A well-known approach to obtain an LP relaxation used for approximation algorithms for Steiner Tree is to first construct the corresponding bidirected graph, in which each undirected edge uv is replaced by the two directed edges uv and vu and setting their weights to the weight of the original undirected edge. Then one of the terminals is declared the *root*, and the constraints of the LP relaxation say that each terminal sends a unit flow to the root. Typically, the LP is formulated using cut constraints, which by the celebrated MaxFlow-MinCut theorem imply that the required flows exist. Roughly speaking, for our integral LP, we formulate the problem by sending a "labelled" unit flow from each terminal to the root. The flows are labelled by pairs of terminal subset $T_1, T_2 \subseteq T$ on each edge e, with the meaning that flows from these two sets converge at e. Because we work with these labelled flows, in contrast to usual flow formulations, we do not obtain an equivalent cut formulation. We present our LP in section 2

Since we use directed edges to formulate flows, in fact we prove Theorem 2 for the more general Directed Steiner Tree problem, where we are given a directed graph $G = (V, E)$ (of which bidirectedness is a special case) with non-negative edge weights $w : E \to \mathbb{R}^+$, a set of *terminals* $T \subseteq V$, and a *root* $r \in V$. The aim is to find an arborescence[2] $A \subseteq G$ of minimum weight, such that there is a path from t to r in A for each $t \in T$.

To prove Theorem 2, in section 3 we develop a primal-dual algorithm for our LP. Interestingly, it turns out that this algorithm can be interpreted as the Dreyfus-Wagner algorithm, and thus we give a polyhedral perspective on this classic algorithm. After proving Theorem 2 we turn to Theorem 1 in section 4. Towards that, we show that the the Steiner Tree polytope contains the Hamiltonian Path polytope as a face, for which exponential lower bounds on the extension complexity can be derived from known lower bounds for Hamiltonian Cycle.

1.2 Related results

Karp [19] mentioned the Steiner Tree problem in his seminal list of NP-hard problems already in 1972, and the problem has since been widely studied. The best approximation algorithm known today is by Byrka et al. [6], and achieves an approximation ratio of $\ln(4) + \epsilon$. Their result uses a hypergraphic LP relaxation, which has also been studied in [7]. Further LPs for Steiner Tree can be found in [15]. Faster FPT algorithms than the one by Dreyfus and Wagner also exist: for arbitrary non-negative edge weights Steiner Tree can be solved in $(2+\epsilon)^k n^{O_\epsilon(1)}$ time [13], and the unweighted problem can even be solved in $2^k n^{O(1)}$ time [25].

[2] a directed graph with exactly one sink of out-degree 0, for which the underlying undirected graph is a tree.

2 The linear program

In this section we describe our linear program and prove its correctness. Throughout this paper we will assume that the root r of the instance is not contained in the terminal set T. For technical reasons, we also have to assume that each non-terminal (so-called *Steiner vertices*) has at most three neighbours, and the root and each terminal has one neighbour. This can be achieved using a standard preprocessing procedure (cf. Appendix A). We call such a preprocessed instance *reduced*.

For any non-empty terminal subset $T' \subseteq T$ we define the *reachability set* $R_{T'} \subseteq V \cup E$ as those vertices $u \in V$ and edges $uv \in E$ for which every terminal $t \in T'$ has a path to u in G. Our *Parameterized Directed Steiner Tree* (PDST) LP has one variable $x_e^{\{T_1,T_2\}}$ for every unordered pair of sets $T_1, T_2 \subseteq T$ that are disjoint, i.e., $T_1 \cap T_2 = \emptyset$, and not both of T_1 and T_2 are empty, i.e., $T_1 \cup T_2 \neq \emptyset$, and for every directed edge $e \in R_{T_1 \cup T_2}$ reachable from both T_1 and T_2. In the following, $\delta^-(v) \subseteq E$ denotes all incoming edges to a vertex v, while $\delta^+(v) \subseteq E$ denotes its outgoing edges.

$$\min \sum_{\substack{T_1 \cup T_2 \neq \emptyset,\\ T_1 \cap T_2 = \emptyset \\ e \in R_{T_1 \cup T_2}}} x_e^{\{T_1,T_2\}} w(e) \quad \text{such that} \tag{PDST}$$

$$\sum_{\substack{T_1 \cup T_2 = T' \\ T_1 \cap T_2 = \emptyset \\ e \in \delta^-(v) \cap R_{T'}}} x_e^{\{T_1,T_2\}} = \sum_{\substack{T'' \subseteq T \setminus T' \\ e \in \delta^+(v) \cap R_{T' \cup T''}}} x_e^{\{T',T''\}} \qquad \begin{cases} \forall T' \subseteq T : T' \neq \emptyset, \text{ and} \\ \forall v \in R_{T'} \setminus (T \cup \{r\}) \end{cases} \tag{1}$$

$$x_e^{\{\{t\},\emptyset\}} = 1 \qquad \{ \forall t \in T, \{e\} = \delta^+(t) \tag{2}$$

$$\sum_{\substack{T_1 \cup T_2 = T' \\ T_1 \cap T_2 = \emptyset}} x_e^{\{T_1,T_2\}} = 0 \qquad \begin{cases} \forall t \in T, \{e\} = \delta^+(t), \text{ and} \\ \forall T' \subseteq T : \emptyset \neq T' \neq \{t\} \end{cases} \tag{3}$$

$$\sum_{\substack{T_1 \cup T_2 = T \\ T_1 \cap T_2 = \emptyset}} x_e^{\{T_1,T_2\}} = 1 \qquad \{ \{e\} = \delta^-(r) \tag{4}$$

$$\sum_{\substack{T_1 \cup T_2 = T' \\ T_1 \cap T_2 = \emptyset}} x_e^{\{T_1,T_2\}} = 0 \qquad \begin{cases} \{e\} = \delta^-(r), \text{ and} \\ \forall T' \subseteq T : \emptyset \neq T' \neq T \end{cases} \tag{5}$$

$$x_e^{\{T_1,T_2\}} \geq 0 \qquad \begin{cases} \forall T_1, T_2 \subseteq T : T_1 \cap T_2 = \emptyset, \\ T_1 \cup T_2 \neq \emptyset, \text{ and} \\ \forall e \in R_{T_1 \cup T_2} \end{cases} \tag{6}$$

Before we prove the correctness of the LP, we try to give some intuition about how it works. As usual, we want an integral solution for the LP to correspond to a solution for the original instance, where exactly one variable corresponding to every edge in the solution is 1. The idea is to look at the solution in the graph as a flow from the terminals to the root. Each terminal pushes a flow of value 1, but as we go closer to the root, the number of edges carrying the flow decrease, while flows from many terminals combine on these edges, but still we would want their corresponding variables to be 1. To this end, we index the variables for the edges with pairs of subsets of T. If there are flows from two subsets of terminals T' and T'' flowing through an edge uv, then we want $x_{uv}^{\{T',T''\}}$ to be set to 1 by a

solution to the LP. From preprocessing, it is guaranteed that in any Steiner arborescence, each vertex has in-degree at most 2. To get the flow for set T' at uv, it must be the case that $T' = T_1 \cup T_2$ for some $T_1, T_2 \subseteq T$ such that $T_1 \cap T_2 = \emptyset$, and T_1 and T_2 had combined to form T' before u. This condition is captured by Constraint (1), which is the crucial constraint of the LP. This process of forming a flow from T' might have happened long before reaching an incoming edge of u and then taking a path to u, so we allow one of T_1 and T_2 to be empty. Constraint (2) ensures that terminals send a flow of 1. Constraint (4) makes sure that there is a flow from all the terminals to the root. Constraint (3) makes sure that a terminal does not send flows for sets other than the singleton set containing itself, and finally Constraint (5) ensures that no other flow from a subset of terminals reaches it, except from the full terminal set. Constraint (5) will not be needed for the correctness of the LP in this section, but will simplify the dual LP given in the next section.

Now we are ready to prove the correctness of the LP, for which we need to argue that there is an integral solution to the LP of cost at most W if and only if there is a Steiner arborescence in the input graph of cost at most W. We begin with the following lemma.

▶ **Lemma 3.** *Let $I := (G, T, r, w)$ be a reduced* Directed Steiner Tree *instance, and let* (PDST) *be the corresponding LP for it. If instance I has a solution of weight at most W, then* (PDST) *has an integral solution of weight at most W.*

Proof. Let A be a minimal solution to I of weight at most W such that every terminal $t \in T$ is a leaf in A and every internal vertex of A has degree at most 3 in A. From minimality of A, we know that A is an arborescence, so for every vertex $v \in V(A) \setminus \{r\}$, we have that $|\delta^+(v) \cap E(A)| = 1$. To get an integral feasible solution for (PDST), we first put $x_e^{\{T_1,T_2\}} := 0$ for all $e \notin E(A) \cap R_{T_1 \cup T_2}$, for all $T_1, T_2 \subseteq T$. We know that every vertex $v \in V(A) \setminus (T \cup \{r\})$, $\deg^+(v) + \deg^-(v) \leq 3$. Now we define a function $T(e)$ for all edges $e = uv \in E(A)$ to be the set of all the terminals in the subtree of A rooted at the vertex u. For every edge $e = uv \in E(A)$ such that $e \neq tv$ for some $t \in T$, u has either one or two incoming edges. If u has only one incoming edge $e' = wu$, then we put $x_e^{\{T(e'),\emptyset\}} := 1$, and $x_e^{\{T_1,T_2\}} := 0$ for all other $T_1, T_2 \subseteq T$. If u has two incoming edges in A, then let $e_1 = w_1u$ and $e_2 := w_2u$ be those two edges. We put $x_e^{\{T(e_1),T(e_2)\}} := 1$ and $x_e^{\{T_1,T_2\}} := 0$ for all other $T_1, T_2 \subseteq T$. For $e \in E(A)$ such that $e = tv$ for some $t \in T$, we put $x_e^{\{\{t\},\emptyset\}} := 1$. For every other variable $x_e^{\{T_1,T_2\}}$, we put $x_e^{\{T_1,T_2\}} := 0$. It is easy to see that this procedure decides to set $x_e^{\{T_1,T_2\}}$ to 1 only if $e \in R_{T_1 \cup T_2}$, and hence the the variable $x_e^{\{T_1,T_2\}}$ exists, and the assignment is valid.

Since for all the edges $e \in E(A)$, exactly one of the variables $x_e^{\{T_1,T_2\}}$ is set to 1 and all other variables are set to 0, it follows that the weight of this assignment for (PDST) is same as weight of A, which is at most W. So all we need to show is that this assignment is feasible for (PDST). It is easy to see that the Constraints (4), (5), and (6) are satisfied by this assignment. To see that Constraint (1) is satisfied, we observe that for any $v \notin V(A)$, all the variables corresponding to edges incident on it are set to 0, and hence (1) is satisfied. For $v \in V(A) \setminus (T \cup \{r\})$, it has exactly one outgoing edge $e = vw$ in A, such that $x_e^{\{T',T''\}} := 1$ for a unique pair $T', T'' \subseteq T$ with $T' \cup T'' \neq \emptyset$ and $T' \cap T'' = \emptyset$. If v has only one incoming edge $e' = uv$ in A, then we have that $T'' = \emptyset$, and T' is the set of terminals in the subtree rooted at u. The assignment puts $x_{e'}^{\{T_1,T_2\}} := 1$ for some $T_1, T_2 \subseteq T'$ such that $T_1 \cap T_2 = \emptyset$ and $T_1 \cup T_2 = T'$, and hence constraint (1) is satisfied. In the other case, let $e_1 = u_1v$ and $e_2 = u_2v$ be the two incoming edges to v in A. Let A_1 and A_2 be subtrees of A rooted at u_1 and u_2 respectively and let T_1 and T_2 be set of terminals in A_1 and A_2. The above assignment puts $x_{e_1}^{\{T_3,T_4\}} = 1$ for some T_3 and T_4 such that T_3 and T_4 are disjoint subsets of T_1 with $T_3 \cup T_4 \neq \emptyset$. Since $x_{e_1}^{\{T_3,T_4\}}$ is the only variable corresponding to e_1 put as 1, we can

see that the Constraint (1) is satisfied for v and T'. Similarly we can show that Constraint (1) is satisfied for v and T'' also. To see that Constraints (2) and (3) are satisfied, we only need to look at the edge $tv \in E(A)$ for $t \in T$. For that, the assignment puts $x_{tv}^{\{\{t\},\emptyset\}} = 1$, and all other variables corresponding to the edge tv are set to 0, so Constraints (2) and (3) are also satisfied. ◀

Next we prove the other direction needed to show the correctness of the LP. This step critically relies on the fact that all edge weights are non-negative.

▶ **Lemma 4.** *Let $I := (G, T, r, w)$ be a reduced* Directed Steiner Tree *instance, and let* (PDST) *be the corresponding LP for it. If* (PDST) *has an integral solution of weight at most W then I has a solution of weight at most W.*

Proof. It suffices to prove the lemma for an integral solution x to (PDST) of minimum weight. Consider the support of x in $G = (V, E)$, i.e., the set $F \subseteq E$ of edges for which $x_e^{\{T_1,T_2\}} > 0$ for some $T_1, T_2 \subseteq T$. We claim that for any terminal $t \in T$ there exists a path from t to the root r in the graph spanned by F in G. Since x is integral, any strictly positive variable has value at least 1. Using the fact that all edge weights are non-negative, this implies that the objective function value of (PDST) is at least the weight of the union of these paths (where each edge is counted only once even if it appears in several paths). As these paths form a feasible solution to the Directed Steiner Tree instance, the lemma follows.

To show the existence of a path from some t to r spanned by edges in F, we show that there is a walk from t to r given by a finite sequence of edges $\{e_i\}_{i\geq 1}$, such that for any $i \geq 1$ the head of e_i is the tail of e_{i+1}, and there exists a set $T_1^i \subseteq T$ with $t \in T_1^i$ and $x_{e_i}^{\{T_1^i,T_2^i\}} > 0$ for some $T_2^i \subseteq T$. By Constraint (2), we have $x_e^{\{\{t\},\emptyset\}} = 1$ for $\{e\} = \delta^+(t)$ and so we may set $e_1 = e$, $T_1^1 = \{t\}$, and $T_2^1 = \emptyset$ for $i = 1$. Now let $i \geq 1$ and assume that we have identified an edge $e_i = uv$ with the properties required for the walk. Note that $v \notin T$, since in the reduced instance G any terminal has in-degree 0. So if $v \neq r$ then $v \in V \setminus (T \cup \{r\})$ and Constraint (1) implies that, if $x_{e_i}^{\{T_1^i,T_2^i\}} > 0$ then there must be an edge $e \in \delta^+(v)$ with $x_e^{\{T',T''\}} > 0$ for $T' = T_1^i \cup T_2^i$ and for some $T'' \subseteq T$. Hence we may set $e_{i+1} = e$, $T_1^{i+1} = T'$, and $T_2^{i+1} = T''$.

This way we obtain a walk of potentially infinite length. Note however that if it is finite then it must end in r, since this is the only condition under which we would not be able to identify the next edge e_i for the sequence in the above argument. To see that the length of the walk is finite, note that $T_1^j \subseteq T_1^i$ for all $i > j$, i.e., the sets T_1^i form a sequence with non-decreasing sizes. Assuming the walk has infinite length, there must thus be some index $j \geq 1$ for which $T_1^i = T_1^j$ for all $i > j$, since every set $T_1^i \subseteq T$ has size at most k. By construction we have $T_1^{i+1} = T_1^i \cup T_2^i$, and so $T_1^i = T_1^j = T_1^{i+1}$ implies $T_2^i = \emptyset$ for all $i > j$. Since there is a finite number of edges in G, the edges e_i with $i > j$ must contain a simple cycle C. We now subtract 1 from each variable $x_e^{\{T_1^j,\emptyset\}}$ where $e \in E(C)$, and claim that the resulting solution x' is a feasible integral solution for (PDST), which however would contradict that x has minimum weight. As C is a simple cycle we subtract 1 from both the left- and right-hand side of Constraint (1) for every vertex v lying on C and $T' = T_1^j$. Constraint (6) is also still valid, since every decreased variable lies in the support of the integral solution x. All other constraints are unchanged, leading to the required contradiction. ◀

Combining Theorem 3 and 4 gives us the following theorem, which proves the correctness of the linear program (PDST). In particular, it implies the first part of Theorem 2, i.e., $ST(G, T) \subseteq \{x \mid \exists y : (x, y) \in Q\}$ where Q is the polyhedron defined by the constraints of (PDST).

▶ **Theorem 5.** *Let $I := (G, T, r, w)$ be a reduced* Directed Steiner Tree *instance, and let* (PDST) *be the corresponding LP for it.* (PDST) *has an integral solution of weight at most W if and only if I has a solution of weight at most W.*

3 Integrality via primal-dual algorithm

In this section we will show that (PDST) is integral. For this we develop a primal-dual algorithm to compute an optimum solution to (PDST), which is also integral. The dual to (PDST) is captured by the following LP, which has a variable $z_v^{T'}$ for every non-empty set $T' \subseteq T$ and vertex $v \in R_{T'}$ reachable from T' corresponding to each of the Constraints (1), (2), (3), (4), and (5). To simplify the notation, we additionally define a variable $z_u^{\emptyset}$ for every vertex $u \in V \setminus T$ and empty terminal set, and set all these to zero. Note that these variables only occur on the right-hand side of Constraint (7) of the dual LP (PDST*) below.

$$\max z_r^T + \sum_{t \in T} z_t^{\{t\}} \text{ such that} \tag{PDST*}$$

$$z_v^{T_1 \cup T_2} \leq z_u^{T_1} + z_u^{T_2} + w(uv) \quad \begin{cases} \forall T_1, T_2 \subseteq T : T_1 \cap T_2 = \emptyset, T_1 \cup T_2 \neq \emptyset \\ \forall uv \in R_{T_1 \cup T_2} : u \notin T \end{cases} \tag{7}$$

$$z_t^{T'} + z_v^{T'} \leq w(tv) \quad \begin{cases} \forall T' \subseteq T : T' \neq \emptyset, \text{ and} \\ \forall t \in T, v \in R_{T'}, tv \in E \end{cases} \tag{8}$$

$$z_u^{\emptyset} = 0 \quad \{ \forall u \in V \setminus T \tag{9}$$

We now describe the algorithm, which uses (PDST*) to construct an integral solution to (PDST). In particular, the algorithm maintains a feasible dual solution to (PDST*) from which in the end it extracts a feasible solution to (PDST).

Initially the algorithm sets all variables $z_v^{T'} = 0$ in (PDST*). Clearly this dual solution is feasible for (PDST*), as all edge weights are non-negative. The algorithm considers all possible non-empty subsets of terminals $T' \subseteq T$ in non-decreasing order of their sizes (breaking ties arbitrarily) to change variables $z_v^{T'}$ where $v \in R_{T'}$. Starting with singleton terminal sets, if $d(u, v)$ denotes the shortest-path distance from u to v in G, for every $t \in T$ and $v \in R_{\{t\}}$ the algorithm sets

$$z_v^{\{t\}} = d(t, v). \tag{10}$$

After this the algorithm considers each $T' \subseteq T$ with $|T'| \geq 2$. For every $v \in R_{T'}$ it sets the variables according to the following recurrence.

$$z_v^{T'} = \min_{\substack{u \in R_{T'} \\ T_1 \cup T_2 = T' \\ T_1 \cap T_2 = \emptyset \\ T_1, T_2 \neq \emptyset}} \{z_u^{T_1} + z_u^{T_2} + d(u, v)\}. \tag{11}$$

Note that here the sets T_1 and T_2 are proper subsets of T' and have thus been considered by the algorithm in a previous step. Also note that u may be equal to v in which case $d(u, v) = 0$. As a consequence, every variable $z_v^{T'}$ is set to a finite value. In particular, if a vertex $u \in R_{T'}$ does not have a path to v then $d(u, v) = \infty$ and so the minimum of the right-hand side of (11) is always obtained by some vertex u that has a path to v.

We remark that setting the variables as shown above can be seen as a step of the dynamic program of the Dreyfus-Wagner algorithm (where the table entries are given by all dual variables).[3] On the other hand, this can also be seen as setting the dual variables to a highest possible value without violating constraints of (PDST*) (as is typical in primal-dual algorithms), where we witness the maximality of the dual values by arguing that certain corresponding constraints of (PDST*) are *tight*, meaning that they are fulfilled with equality. This is formalized by the following two lemmas, where we first show that the dual solution is feasible.

▶ **Lemma 6.** *Given a non-empty terminal subset $T' \subseteq T$, after the algorithm sets all dual variables $z_v^{T'}$ with $v \in R_{T'}$ according to either* (10) *or* (11)*, the dual solution is feasible for* (PDST*)*.*

Proof. Neither (10) nor (11) change the value of any variable $z_u^{\emptyset}$, and as these are set to 0 initially, Constraint (9) is never violated. For any $t \in T$ and $T' \subseteq T$ such that $|T'| \geq 2$, the algorithm never sets $z_t^{T'}$ to a non-zero value because t is a leaf and cannot be reached by all the terminals in T'. Constraint (8) can only be violated for $T' \subseteq T$ of size at least 2 if the algorithm sets $z_v^{T'}$ to a non-zero value according to (11). This is not possible because v is the only neighbour of t and so there does not exist any $u \in R_{T'}$ different from v. So, Constraint (8) can only be violated if the algorithm sets a dual variable $z_v^{\{t\}}$ according to (10), since $|T'| \geq 2$ in (11). However, $z_t^{\{t\}} = d(t,t) = 0$, and so a variable $z_v^{\{t\}}$ is set to its maximum possible value without violating Constraint (8), i.e., it is tight.

For Constraint (7) and an edge uv, note that if $T_1 \neq \emptyset$ and $T_2 \neq \emptyset$ then (11) sets $z_v^{T_1 \cup T_2}$ to a value of at most $z_u^{T_1} + z_u^{T_2} + d(u,v)$, since the shortest path from u to v is considered in the right-hand side of (11). Because $d(u,v) \leq w(uv)$, Constraint (7) is not violated in this case. Furthermore, if one of the two sets, say T_2, is empty, then $z_u^{T_2} = 0$ by Constraint (9) and so Constraint (7) is given by $z_v^{T_1} \leq z_u^{T_1} + w(uv)$. According to (11), $z_u^{T_1}$ is set to some value $z_q^{T_1'} + z_q^{T_2'} + d(q,u)$ for some vertex q and sets T_1', T_2', which together form T_1. The shortest path from q to v is at most as long as the shortest path from q to u plus the edge uv, i.e., $d(q,v) \leq d(q,u) + w(uv)$. Thus according to (11), $z_v^{T_1}$ is set to a value of at most $z_q^{T_1'} + z_q^{T_2'} + d(q,v) \leq z_q^{T_1'} + z_q^{T_2'} + d(q,u) + w(uv) = z_u^{T_1} + w(uv)$, which concludes the proof. ◀

To identify the tight constraints after running the algorithm, we say that Constraint (7) of (PDST*) is *tight for* $(uv, \{T_1, T_2\})$ if the corresponding inequality is tight, i.e., $z_v^{T_1 \cup T_2} = z_u^{T_1} + z_u^{T_2} + w(uv)$, and Constraint (8) of (PDST*) is *tight for* (tv, T') if $z_t^{T'} + z_v^{T'} = w(tv)$ for the corresponding inequality. Given $t \in T$ and $v \in R_{\{t\}}$ so that $z_v^{\{t\}}$ is set according to (10), we say that a $t \to v$ path P in G is *tight for* $\{t\}$ if

- for the first edge tv' of P, Constraint (8) is tight for $(tv', \{t\})$, and
- for every edge $u'v'$ of P, Constraint (7) is tight for $(u'v', \{\{t\}, \emptyset\})$.

Given $|T'| \geq 2$ and $v \in R_{T'}$ so that $z_v^{T'}$ is set according to (11) where the minimum of the right-hand side of (11) is obtained by $u \in R_{T'}$ and $T_1, T_2 \subseteq T'$, we say that a $u \to v$ path P in G is *tight for* $\{T_1, T_2\}$ if

- for the first edge uv' of P, Constraint (7) is tight for $(uv', \{T_1, T_2\})$, and
- for every edge $u'v'$ of P, Constraint (7) is tight for $(u'v', \{T', \emptyset\})$.

[3] Note that the dynamic program of Dreyfus and Wagner is typically set up slightly differently, such that the table entries are initially set to ∞. This implies that for any vertex v that is not reachable from a terminal set T', the corresponding entry for v and T' will be set to ∞ when filling the table. Here we instead initialize the dual variables to 0 in order to obtain a feasible dual solution. We then counteract any effects that this has on the recursion by only considering vertices in the reachability set when changing variables.

▶ **Lemma 7.** *Given a terminal $t \in T$, after the algorithm sets all dual variables $z_v^{\{t\}}$ with $v \in R_{\{t\}}$ according to (10), any shortest $t \to v$ path for $v \in R_{\{t\}}$ is tight for $\{t\}$. Given a terminal subset $T' \subseteq T$ with $|T'| \geq 2$, after the algorithm sets all dual variables $z_v^{T'}$ with $v \in R_{T'}$ according to (11) such that $z_v^{T'} = z_u^{T_1} + z_u^{T_2} + d(u,v)$ for some $u \in R_{T'}$ and $T_1, T_2 \subseteq T'$, any shortest $u \to v$ path for $v \in R_{T'}$ is tight for $\{T_1, T_2\}$.*

Proof. Consider a terminal $t \in T$, $v \in R_{\{t\}}$, and any shortest $t \to v$ path P. We saw in the proof of Lemma 6 above that for the first edge tv' of P, Constraint (8) is tight for $(tv', \{t\})$. Now consider any edge $u'v'$ of P. According to (10), $z_{v'}^{\{t\}} = d(t,v')$ and $z_{u'}^{\{t\}} = d(t,u')$. As u' and v' lie on the shortest path P, u' also lies on a shortest $t \to v'$ path and so $d(t,v') = d(t,u') + w(u'v')$. Hence we get $z_{v'}^{\{t\}} = z_{u'}^{\{t\}} + w(u'v')$, i.e., Constraint (7) is tight for $(u'v', \{\{t\}, \emptyset\})$ using that $z_{u'}^{\emptyset} = 0$ according to Constraint (9).

Given a terminal set T' with $|T'| \geq 2$ and $v \in R_{T'}$, let $z_v^{T'}$ be set according to (11) so that $z_v^{T'} = z_u^{T_1} + z_u^{T_2} + d(u,v)$ for some $u \in R_{T'}$ and $T_1, T_2 \subseteq T'$. Consider any vertex p of any shortest $u \to v$ path P. According to (11), $z_p^{T'} = z_q^{T_1'} + z_q^{T_2'} + d(q,p)$ for some $q \in R_{T'}$ and $T_1', T_2' \subseteq T'$, which together form T'. As $z_u^{T_1} + z_u^{T_2} + d(u,p)$ is considered by (11) when setting $z_p^{T'}$, we have $z_q^{T_1'} + z_q^{T_2'} + d(q,p) \leq z_u^{T_1} + z_u^{T_2} + d(u,p)$. Since p lies on the path P leading to v, there exists a path from q to v via p, and $d(q,v) \leq d(q,p) + d(p,v)$. Consequently,

$$\begin{aligned} z_q^{T_1'} + z_q^{T_2'} + d(q,v) &\leq z_q^{T_1'} + z_q^{T_2'} + d(q,p) + d(p,v) \\ &\leq z_u^{T_1} + z_u^{T_2} + d(u,p) + d(p,v) \\ &= z_u^{T_1} + z_u^{T_2} + d(u,v), \end{aligned}$$

where the last equality uses the fact that p lies on the shortest $u \to v$ path P. The value $z_q^{T_1'} + z_q^{T_2'} + d(q,v)$ is considered when setting $z_v^{T'}$ according to (11). But since $z_u^{T_1} + z_u^{T_2} + d(u,v)$ minimizes the right-hand side of (11) when setting $z_v^{T'}$, the latter inequalities are in fact equalities. In particular, after subtracting $d(p,v)$ from the above, the second (in)equality implies $z_q^{T_1'} + z_q^{T_2'} + d(q,p) = z_u^{T_1} + z_u^{T_2} + d(u,p)$. As $z_p^{T'} = z_q^{T_1'} + z_q^{T_2'} + d(q,p)$ this means that in fact $z_p^{T'} = z_u^{T_1} + z_u^{T_2} + d(u,p)$.

Now consider the first edge uv' of the shortest $u \to v$ path P, for which $w(uv') = d(u,v')$. By setting $p = v'$, from our previous conclusion we obtain $z_{v'}^{T'} = z_u^{T_1} + z_u^{T_2} + w(uv')$, i.e., Constraint (7) is tight for $(uv', \{T_1, T_2\})$. Finally, consider any edge $u'v'$ of P. Since p was an arbitrary vertex of P in the above argument, we can conclude that both $z_{v'}^{T'} = z_u^{T_1} + z_u^{T_2} + d(u,v')$ and $z_{u'}^{T'} = z_u^{T_1} + z_u^{T_2} + d(u,u')$. From $d(u,v') = d(u,u') + w(u'v')$ we thus get $z_{v'}^{T'} = z_{u'}^{T'} + w(u'v')$, i.e., Constraint (7) is tight for $(u'v', \{T', \emptyset\})$. ◀

We now use the tight paths identified by Theorem 7 to extract a feasible integral primal solution to (PDST) from the dual solution computed by the algorithm. This can be done by the following recursive procedure, for which we initially set all variables $x_e^{\{T_1,T_2\}} = 0$. Each recursive call is given by a function $\textsc{primSol}(v, T')$ on some vertex v and terminal set T' with $v \in R_{T'}$, and we begin the recursion with a call $\textsc{primSol}(r, T)$ on the root r and the full terminal set T. Note that w.l.o.g., we may assume that every terminal can reach the root, i.e., $r \in R_T$. We will maintain the invariant that $v \in R_{T'}$ during each recursive call.

Given a vertex v and a terminal set T', the function $\textsc{primSol}(v, T')$ does the following. If $|T'| \geq 2$ then let $P_{T'}$ be any tight shortest $u \to v$ path for $\{T_1, T_2\}$ given by Theorem 7 (using that $v \in R_{T'}$) for the vertex $u \in R_{T'}$ and sets $T_1, T_2 \subseteq T'$ for which the minimum is obtained on the right-hand side of (11) when setting $z_v^{T'}$. If f denotes the first edge of $P_{T'}$, then in (PDST) the function sets $x_f^{\{T_1,T_2\}} = 1$, and $x_e^{\{T',\emptyset\}} = 1$ for every edge $e \neq f$ of $P_{T'}$. After this the function makes a recursive call $\textsc{primSol}(u, T_1)$ on u and T_1, and thereafter a call $\textsc{primSol}(u, T_2)$ on u and T_2. Note that in particular $u \in R_{T_1}$ and $u \in R_{T_2}$ and thus the invariant $v \in R_{T'}$ is given for each recursive call.

Finally, if $T' = \{t\}$ for some terminal $t \in T$, then let $P_{\{t\}}$ be any tight shortest $t \to v$ path for $\{t\}$ given by Lemma x7 (using that $v \in R_{\{t\}}$). The function PRIMSOL$(v, \{t\})$ sets $x_e^{\{\{t\},\emptyset\}} = 1$ in (PDST) for every edge e of $P_{\{t\}}$. Here the recursion ends.

Note that, since (PDST) only has equality constraints, using tight paths to set variables of the primal solution to 1 means that the primal and dual solutions are *complementary slack*, i.e., for every non-zero variable of the primal (dual) LP the corresponding constraint in the dual (primal) LP is tight. It is well-known (cf. [21]) that feasible primal and dual solutions are complementary slack if and only if both solutions are optimal for their LPs. As we prove in the next lemma, the above procedure computes a feasible integral primal solution. By Theorem 6 also the dual solution is feasible, and so the primal and dual solutions are optimal for (PDST) and (PDST*), respectively. In particular, the primal solution is an optimal integral $(0, 1)$-solution to (PDST), proving the second part of Theorem 2.

▶ **Lemma 8.** *The above recursive procedure computes a feasible integral primal solution to* (PDST) *for any reduced* DIRECTED STEINER TREE *instance.*

Proof. First observe that for any terminal subset T', (11) always minimizes over proper subsets $T_1, T_2 \subset T'$ partitioning T'. Since the recursive calls are on the same sets T_1 and T_2, this implies that at most one recursive call is made on any set T', and in particular, the path $P_{T'}$ is well-defined. Moreover, for any T' the procedure only sets variables $x_e^{\{T_1,T_2\}}$ to non-zero values if $T_1 \cup T_2 = T'$ (where possibly $T_1 = T'$ and $T_2 = \emptyset$). As a consequence, for every non-zero variable $x_e^{\{T_1,T_2\}}$ with $T_1 \cup T_2 = T'$, e lies on the path $P_{T'}$.

Now consider Constraint (1) of (PDST) for some vertex $v \in V \setminus (T \cup \{r\})$ and non-empty terminal subset $T' \subseteq T$. If any variable $x_e^{\{T_1,T_2\}}$ contributes a non-zero value to the left-hand side of this constraint, then e lies on $P_{T'}$, since $T_1 \cup T_2 = T'$ by definition of Constraint (1). Note that there can be only one non-zero variable on the left-hand side of the constraint, as there is only one recursive call on T'. Furthermore, v can only be the last vertex or an internal vertex of $P_{T'}$, since $e \in \delta^-(v)$ for any left-hand side variable.

If v is the last vertex of the tight path $P_{T'}$ for $\{T_1, T_2\}$, then the procedure did a recursive call PRIMSOL(v, T') on T' and v. Since $v \neq r$, this call was made due to a previous call on a set $T' \cup T''$ for some non-empty set T'', and v is the first vertex of the tight path $P_{T' \cup T''}$ for $\{T', T''\}$. During this call the function sets the variable $x_f^{\{T',T''\}} = 1$ for the the first edge $f \in \delta^+(v)$ of $P_{T' \cup T''}$. This variable thus contributes a non-zero value to the right-hand side of the constraint for v and T'. As v is the last vertex of $P_{T'}$ and only the unique recursive call on T' sets variables $x_e^{\{T',\emptyset\}}$ to non-zero values, all such variables with $e \in \delta^+(v)$ on the right-hand side of the constraint are set to zero. Furthermore, for each non-zero variable $x_e^{\{T',T''\}}$ with $T'' \neq \emptyset$ and $e \in \delta^+(v)$, there is a recursive call PRIMSOL(v, T') on v and T': the procedure will only set such a variable to a non-zero value if both T' and T'' are non-empty and e is the first edge of the tight path for $\{T', T''\}$. As each recursive call is on a unique set T', this means that there can only be one non-zero variable on the right-hand side of the constraint. Since all non-zero variables are set to 1, if the left-hand side has one variable set to 1, then so does the right-hand side. Note that the above arguments also imply the reverse: if the right-hand side has a variable $x_f^{\{T',T''\}} = 1$ with $f \in \delta^+(v)$, then the recursive call PRIMSOL(v, T') sets a variable $x_e^{\{T_1,T_2\}} = 1$ with $e \in \delta^-(v)$, and there can be only one such non-zero variable on each of the right- and left-hand sides. Therefore the constraint is valid if v is the last vertex of $P_{T'}$.

If v is an internal vertex of $P_{T'}$, the procedure sets $x_e^{\{T',\emptyset\}}$ for the edge $e \in \delta^+(v)$ lying on $P_{T'}$ to a non-zero value. As only the unique call on T' sets variables $x_e^{\{T',\emptyset\}}$, no other such variable with $e \in \delta^+(v)$ is set to a non-zero value. Now consider a variable $x_f^{\{T',T''\}}$

with $f \in \delta^+(v)$ and $T'' \neq \emptyset$. Since both T' and T'' are non-empty, if $x_f^{\{T',T''\}}$ is non-zero then f must be the first edge of a tight path $P_{T' \cup T''}$ and $x_f^{\{T',T''\}}$ is set during a recursive call on $T' \cup T''$. However, since $|T' \cup T''| \geq 2$, this would mean during this call a recursive call PRIMSOL(v, T') was made on v and T', implying that v is the last vertex of $P_{T'}$ – a contradiction. Hence as before, if the left-hand side of the constraint has one variable set to 1, then so does the right-hand side, and the reverse is also true. Thus Constraint (1) is valid.

For the other constraints of (PDST) it is easy to see that they are valid: for Constraint (2) the procedure sets $x_e^{\{\{t\},\emptyset\}} = 1$ for $\{e\} \in \delta^+(t)$, as the recursion begins with the full terminal set T and so must end in each terminal $t \in T$, and the tight path $P_{\{t\}}$ for $\{t\}$ always has t as the first vertex. Since all variables are initially set to zero, for Constraint (3) it suffices to note that no tight path $P_{T'}$ with $T' \neq \{t\}$ can contain terminal t, as the DIRECTED STEINER TREE instance is reduced and so every terminal has only one neighbour. Constraint (4) is valid since the recursion begins at the root r with the full terminal set T, so that some variable $x_e^{\{T_1,T_2\}}$ with $T_1 \cup T_2 = T$ is set to 1. As the recursive call on T is unique, exactly one such variable is non-zero. For Constraint (5) it again suffices to note that no tight path $P_{T'}$ with $T' \neq T$ can contain r, since also the root only has one neighbour. Finally, Constraint (6) is obviously valid, since the procedure sets all variables to either 0 or 1. ◀

4 Extension complexity of the Steiner tree polytope

In this section we prove Theorem 1, i.e., that the extension complexity of the Steiner tree polytope $ST(G,T)$ is exponential. For that, we will need the following two additional polytopes. One is for the HAMILTONIAN PATH problem, where we are given an undirected graph $G = (V, E)$ and two vertices $u, v \in V$ and we need to decide whether there exists a path P in G between u and v such that P visits all vertices of G. A related problem is the HAMILTONIAN CYCLE problem, where we need to decide whether an undirected graph G has a cycle that visits all the vertices of G.

$$HP(G,u,v) := \text{conv.hull } \{x^{E(P)} \in \{0,1\}^{|E(G)|} \mid P \text{ is a Hamiltonian } u \to v \text{ path in } G\}$$
$$HC(G) := \text{conv.hull } \{x^{E(C)} \in \{0,1\}^{|E(G)|} \mid C \text{ is a Hamiltonian cycle in } G\}$$

Now, we first show a lemma that relates the extension complexity of $HP(G,u,v)$ and $ST(G,T)$ for any graph G. In fact, we will show the lemma for the case when the number of terminals for the STEINER TREE instance is 2. In the following we denote the extension complexity of any polytope Q by $xc(Q)$.

▶ **Lemma 9.** *For any graph G, $xc(HP(G,u,v)) \leq xc(ST(G,\{u,v\}))$.*

Proof. Observe that $\sum_{e \in E(G)} x_e \leq n-1$ is a valid inequality for $ST(G,T)$, as any Steiner tree has at most $n-1$ edges, and thus $ST(G,T) \cap \{x \mid \sum_{e \in E} x_e = n-1\}$ is a face of $ST(G,T)$. We want to show that $HP(G,u,v) = ST(G,\{u,v\}) \cap \{x \mid \sum_{e \in E} x_e = n-1\}$. This would mean that the polytope $HP(G,u,v)$ is a face of the polytope $ST(G,\{u,v\})$, which would imply that the extension complexity of $HP(G,u,v)$ is at most the extension complexity of $ST(G,\{u,v\})$, and thus prove the lemma.

Now, to show the claim, it is enough to show that the extreme points of the left-hand side belong to the polytope on the right-hand side and vice versa. We look at an extreme point $x \in HP(G,u,v)$, that is, there exists a Hamiltonian path P between u and v such that $x = x^{E(P)}$. Clearly, P has $n-1$ edges, so we have that $\sum_{e \in E} x_e = n-1$. On the other hand, since P is a path between u and v, it is a minimal connected subgraph containing u and v and hence $x \in ST(G,\{u,v\})$.

For the other direction, let us take an extreme point of the polytope $ST(G, \{u, v\}) \cap \{x \mid \sum_{e \in E} x_e = n - 1\}$. As observed earlier, for all the extreme points x of $ST(G, \{u, v\})$, $\sum_{e \in E} x_e \leq n - 1$ is a valid inequality. This means that the polytope $ST(G, \{u, v\})$ lies on one side of the hyperplane $\{x \mid \sum_{e \in E} x_e = n - 1\}$. This gives us that all the extreme points of $ST(G, \{u, v\}) \cap \{x \mid \sum_{e \in E} x_e = n - 1\}$ are also extreme points of $ST(G, \{u, v\})$. So, to prove the other direction, it is enough to look at the extreme points of $ST(G, \{u, v\})$. Now, let x be an extreme point of $ST(G, \{u, v\})$, which also belongs to $\{x \mid \sum_{e \in E} x_e = n - 1\}$. This means that $x = x^{E(H)}$ for some Steiner tree H for $(G, \{u, v\})$. We also know that $\sum_{e \in E} x_e = n - 1$. Since H is a minimal subgraph connecting u and v having $n - 1$ edges, it has to be a Hamiltonian path between u and v, and hence $x \in HP(G, u, v)$. This finishes the proof of the lemma. ◀

Now we state the following lemma by Balas [1, 2], which bounds the extension complexity of union of several polytopes.

▶ **Theorem 10** ([1, 2]). *Consider q polytopes $P^i \subseteq \mathbb{R}^n$, $i = 1, \ldots, q$ and write $P :=$ conv.hull$(\cup_{i \in [q]} P^i)$. Then, $xc(P) \leq q + \sum_{i \in [q]} xc(P^i)$.*

We use this theorem in the following to relate the extension complexities of Hamiltonian cycles of the complete graph K_n and Hamiltonian paths.

▶ **Lemma 11.** *Let K_n be a complete graph with n vertices and u and v be two arbitrary vertices of K_n. Then $xc(HC(K_n)) \leq \binom{n}{2} + \binom{n}{2} \cdot xc(HP(K_n - uv, u, v))$, where $K_n - uv$ is the graph K_n with the edge uv removed.*

Proof. We will show $HC(K_n)$ to be convex hull of the union of $\binom{n}{2}$ polytopes, which have their extension complexity same as that of $HP(K_n - uv, u, v)$. We take $Q_{uv} := HP(K_n - uv, u, v)$ for all $uv \in E(K_n)$. Observe that the dimension of Q_{uv} is $\binom{n}{2} - 1$. For each $x \in Q_{uv}$, we make a $y \in P_{uv}$ of length $\binom{n}{2}$ by adding the co-ordinate for the edge uv and giving it value 1. Since P_{uv} is just an embedding of the $(\binom{n}{2} - 1)$-dimensional polytope Q_{uv} into $\binom{n}{2}$-dimensional space by fixing the additional co-ordinate to be 1, the number of facets of P_{uv} and Q_{uv} remain the same, which is at most $xc(HP(K_n - uv, u, v))$. Observe that for all edges uv and $u'v'$ in $E(K_n)$, the polytopes $HP(K_n - uv, u, v)$ and $HP(K_n - u'v', u', v')$ are isomorphic to each other and hence have the same extension complexity. So the polytopes P_{uv} and $P_{u'v'}$ also have the same extension complexity for all $uv, u'v' \in E(K')$.

Now we just need to show that $HC(K_n) = \text{conv.hull}(\cup_{uv \in E(K_n)} P_{uv})$, and the proof of the lemma will follow by Theorem 10. Let us look at an extreme point $x \in HC(K_n)$. Clearly, $x = x^{E(C)}$ for some Hamiltonian cycle C of K_n. Let $e = uv$ be an arbitrary edge in C. We claim that $x \in P_{uv}$. For that, first observe that C contains a Hamiltonian path between u and v, which is the cycle C with edge uv removed. So the characteristic vector of C is the same as the characteristic vector of $C - uv$ in $K_n - uv$ along with 1 at the co-ordinate for the edge uv, so $x \in P_{uv}$. For the other side, let x be an extreme point in $\cup_{uv \in E(K_n)} P_{uv}$ and hence in some P_{uv}. From the construction of P_{uv}, x has the coordinates for $E(P)$ and the edge uv as 1, where P is some Hamiltonian path between u and v in $K_n - uv$. This shows that x is a characteristic vector of a Hamiltonian cycle in K_n and hence $x \in HC(K_n)$. ◀

To show Theorem 1 with the help of the lemmas proved above, we use the following result about the extension complexity of Hamiltonian cycles in complete graphs.

▶ **Theorem 12** ([26]). $xc(HC(K_n)) = 2^{\Omega(n)}$.

Proof of Theorem 1. For the sake of contradiction, let us assume that Theorem 1 is not true. This means that for all constants $c > 0$, there exists a function f such that for all n vertex graphs G, $xc(ST(G,T)) < f(k)2^{cn}$, where $k = |T|$. Now, Lemma 9 and Lemma 11 imply that $xc(HC(K_n)) \leq \binom{n}{2} + \binom{n}{2} xc(HP(K_n - uv, u, v)) \leq \binom{n}{2} + \binom{n}{2} xc(ST(K_n - uv, \{u, v\}))$. We know that $xc(ST(K_n - uv, \{u, v\})) < f(2)2^{cn}$ for some function f and all constants c. This would mean that $xc(HC(K_n)) < d2^{cn}$ for all constants c, where $d = f(2)$ is another constant, which contradicts Theorem 12. This finishes the proof of Theorem 1. ◀

References

1 Egon Balas. Disjunctive programming and a hierarchy of relaxations for discrete optimization problems. *SIAM Journal on Algebraic Discrete Methods*, 6(3):466–486, 1985.

2 Egon Balas. Disjunctive programming: Properties of the convex hull of feasible points. *Discrete Applied Mathematics*, 89(1-3):3–44, 1998.

3 Francisco Barahona. On cuts and matchings in planar graphs. *Mathematical Programming*, 60(1):53–68, June 1993. doi:10.1007/BF01580600.

4 Austin Buchanan. Extended formulations for vertex cover. *Operations Research Letters*, 44(3):374–378, 2016.

5 Austin Loyd Buchanan. *Parameterized Approaches for Large-Scale Optimization Problems*. PhD thesis, 2015.

6 J. Byrka, F. Grandoni, T. Rothvoß, and Laura Sanità. An improved LP-based approximation for Steiner tree. In *Proc. 42nd STOC*, pages 583–592, 2010. doi:10.1145/1806689.1806769.

7 Deeparnab Chakrabarty, Jochen Könemann, and David Pritchard. Hypergraphic lp relaxations for steiner trees. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 383–396. Springer, 2010.

8 Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theor. Comput. Sci.*, 411(40-42):3736–3756, 2010.

9 Jun-Dong Cho. *Steiner Tree Problems in VLSI Layout Designs*, pages 101–173. Springer US, Boston, MA, 2001.

10 Michele Conforti, Gérard Cornuéjols, and Giacomo Zambelli. Extended formulations in combinatorial optimization. *Annals of Operations Research*, 204(1):97–143, 2013.

11 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.

12 S. E. Dreyfus and R. A. Wagner. The steiner problem in graphs. *Networks*, 1(3):195–207, 1971.

13 Bernhard Fuchs, Walter Kern, D Molle, Stefan Richter, Peter Rossmanith, and Xinhui Wang. Dynamic programming for minimum steiner trees. *Theory of Computing Systems*, 41(3):493–500, 2007.

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

15 Michel X Goemans and Young-Soo Myung. A catalog of steiner tree formulations. *Networks*, 23(1):19–28, 1993.

16 F.K. Hwang, D.S. Richards, and P. Winter. *The Steiner Tree Problem*. ISSN. Elsevier Science, 1992.

17 A.B. Kahng and G. Robins. *On Optimal Interconnections for VLSI*. The Springer International in Engineering and Computer Science. Springer US, 2013.

18 V Kaibel. Extended formulations in combinatorial optimization. *Optima 85*, page 14. URL: http://www.mathopt.org/Optima-Issues/optima85.pdf.

19 R. M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972.

20 Petr Kolman, Martin Koutecký, and Hans Raj Tiwary. Extension complexity, mso logic, and treewidth. In *15th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2016.

21 Bernhard Korte and Jens Vygen. *Combinatorial optimization*, volume 2. Springer.

22 B.H. Korte. *Paths, flows, and VLSI-layout*. Algorithms and combinatorics. Springer-Verlag, 1990.

23 R. Kipp Martin. Using separation algorithms to generate mixed integer model reformulations. *Oper. Res. Lett.*, 10(3):119–128, 1991. doi:10.1016/0167-6377(91)90028-N.

24 R. Kipp Martin, Ronald L. Rardin, and Brian A. Campbell. Polyhedral characterization of discrete dynamic programming. *Operations Research*, 38(1):127–138, 1990. URL: http://www.jstor.org/stable/171304.

25 Jesper Nederlof. Fast polynomial-space algorithms using inclusion-exclusion. *Algorithmica*, 65(4):868–884, 2013.

26 Thomas Rothvoss. The matching polytope has exponential extension complexity. *J. ACM*, 64(6):41:1–41:19, 2017.

27 Matias Siebert, Shabbir Ahmed, and George Nemhauser. A linear programming based approach to the steiner tree problem with a fixed number of terminals. *Networks*, n/a(n/a).

28 François Vanderbeck and Laurence A. Wolsey. *Reformulation and Decomposition of Integer Programs*, pages 431–502. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

A Preprocessing

In this section, given an instance (G, T, r, w) of Directed Steiner Tree, we will transform it into an equivalent instance (G', T', r', w'), where G' is a directed graph and G has a solution of weight W if and only if there exists a solution for G' of weight W where all the terminals are leaves with out-degree 1, the root is a leaf with in-degree 1 and for every other vertex v, we have that $\deg^+(v) + \deg^-(v) \leq 3$. Formally, we want to prove the following lemma.

▶ **Lemma 13.** *Given an instance (G, T, r, w) of Directed Steiner Tree, in polynomial time, we can transform it into another Directed Steiner Tree instance (G', T', r', w') such that (G, T, r, w) has a solution A of weight W if and only if (G', T', r', w') has a solution A' of weight W where all the terminals are leaves with out-degree 1, the root is a leaf with in-degree 1, and for all $v \in V(G') \setminus (T' \cup \{r'\})$, we have $\deg^+(v) + \deg^-(v) \leq 3$.*

Proof. We will prove the lemma in two stages. First we transform (G, T, r, w) to (G_1, T_1, r_1, w_1) where G_1 is the graph obtained by making a copy t' of every terminal $t \in T$, and adding the edge (t', t) with zero weight. Let T^* be the set of these new vertices. We also add a vertex r^* as a new root, and add the edge (r, r^*) with zero weight. Then we take $T_1 := T^*$, $r_1 := r^*$, and w_1 is obtained by adding the new zero edge weights to w. It is easy to see that (G, T, r, w) has a solution of weight W if and only if (G_1, T_1, r_1, w_1) has a solution of weight W where all the terminals are leaves with out-degree 1 and the root is a leaf with in-degree 1.

Given (G_1, T_1, r_1, w_1) of Directed Steiner Tree as described above, we make an instance (G_2, T_2, r_2, w_2) of Directed Steiner Tree by the following transformation. For every vertex $v \in V(G_1) \setminus (T_1 \cup \{r_1\})$ with in-degree $\deg^-(v)$ and out-degree $\deg^+(v)$ such that $\deg^+(v) + \deg^-(v) \geq 4$, we replace it with a cycle of length $d := \deg^+(v) + \deg^-(v)$ defined as $C_v = v_1 v_2 \dots v_d v_1$ where $v_1, v_2, \dots, v_d$ are new vertices. Then for every neighbour u of v, if $uv \in E(G_1)$ (or $vu \in E(G_1)$), we find a unique v_i of v, a unique copy u_j of u, and add the edge $v_i u_j$ (or $u_j v_i$). Then we take $T_2 := T_1$ and $r_2 := r_1$ This way, the graph G_2 has maximum degree 3 and all the terminals are leaves in G_2 with out-degree 1 and the root is a

leaf with in-degree 1. For every edge e in the cycle C_v, we put $w_2(e) := 0$. For every other edge $u_iv_j \in E(G_2)$ such that u_i and v_j are in cycles C_u and C_v corresponding to $u \in V(G_1)$ and $v \in V(G_1)$, we put $w_2(u_iv_j) := w_1(uv)$. Given a solution of weight W of (G_2, T_2, r_2, w_2), we can transform it into a solution of weight W for (G_1, T_1, r_1, w_1) by contracting the zero weight edges of the cycles in the solution. On the other hand, given a solution of weight W for (G_1, T_1, r_1, w_1), for an edge uv in the solution, we pick the edge between unique copies u_i and v_j of u and v corresponding to the edge uv and connect all these edges by picking as many edges from C_u and C_v as we want at no extra cost. Observe that the terminals and the root are leaves in G_2, so they will be leaves in any solution as well. This shows that (G_1, T_1, r_1, w_1) has a solution of weight W where all the terminals are leaves with out-degree 1 and the root is also a leaf with in-degree 1 if and only if (G_2, T_2, r_2, w_2) has a solution of weight W where all the terminals are leaves with out-degree 1 and the root is also a leaf with in-degree 1 and all the vertices $v \in V(G_2) \setminus (T_2 \cup \{r_2\})$, we have that $\deg^+(v) + \deg^-(v) \leq 3$. Finally taking $(G', T', r', w') := (G_2, T_2, r_2, w_2)$ finishes the proof of the lemma. ◀

An Investigation of the Recoverable Robust Assignment Problem

Dennis Fischer
Department of Computer Science, RWTH Aachen, Germany

Tim A. Hartmann
Department of Computer Science, RWTH Aachen, Germany

Stefan Lendl
Institute of Operations und Information Systems, Universität Graz, Austria

Gerhard J. Woeginger
Department of Computer Science, RWTH Aachen, Germany

Abstract

We investigate the so-called recoverable robust assignment problem on complete bipartite graphs, a mainstream problem in robust optimization: For two given linear cost functions c_1 and c_2 on the edges and a given integer k, the goal is to find two perfect matchings M_1 and M_2 that minimize the objective value $c_1(M_1) + c_2(M_2)$, subject to the constraint that M_1 and M_2 have at least k edges in common.

We derive a variety of results on this problem. First, we show that the problem is W[1]-hard with respect to parameter k, and also with respect to the complementary parameter $k' = n/2 - k$. This hardness result holds even in the highly restricted special case where both cost functions c_1 and c_2 only take the values 0 and 1. (On the other hand, containment of the problem in XP is straightforward to see.) Next, as a positive result we construct a polynomial time algorithm for the special case where one cost function is Monge, whereas the other one is Anti-Monge. Finally, we study the variant where matching M_1 is frozen, and where the optimization goal is to compute the best corresponding matching M_2. This problem variant is known to be contained in the randomized parallel complexity class RNC[21], and we show that it is at least as hard as the infamous problem Exact Red-Blue Matching in Bipartite Graphs whose computational complexity is a long-standing open problem.

2012 ACM Subject Classification Mathematics of computing → Graph theory; Theory of computation → Fixed parameter tractability

Keywords and phrases assignment problem, matchings, exact matching, robust optimization, fixed paramter tractablity, RNC

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.19

Funding Stefan Lendl acknowledges the support of the Austrian Science Fund (FWF): W1230. Dennis Fischer and Gerhard J. Woeginger have been supported by the DFG RTG 2236 "UnRAVeL".

Acknowledgements We thank Bettina Klinz for several helpful discussions about the topic.

1 Introduction

The Assignment Problem (AP) is a fundamental and well-investigated problem in discrete optimization: For the complete bipartite graph $K_{n,n} = (V, E_{n,n})$ with given costs $c : E_{n,n} \to \mathbb{R}$ on the edges, the AP asks for a perfect matching M in $K_{n,n}$ that minimizes the total cost $c(M)$. The AP can be solved in polynomial time, by using for instance the Hungarian method or techniques from network flow theory; see Burkard, Dell'Amico & Martello [3].

[1] RNC[2] is the randomized version of NC[2]. For a definition of NC[2] see [1].

In this paper we study a variant of the AP from the area of robust optimization, which we denote as Recoverable Assignment Problem (RecovAP). An instance of RecovAP consists of two cost functions $c_1, c_2 : E_{n,n} \to \mathbb{R}$ on the edges of $K_{n,n}$ together with an integer bound k. The goal is to find two perfect matchings M_1 and M_2 that minimize the objective value $c_1(M_1) + c_2(M_2)$, subject to the constraint that M_1 and M_2 have at least k edges in common. We also consider the following two non-trivial special cases of RecovAP:

- Consider an arbitrary (bipartite) subgraph $G = (V, E)$ of $K_{n,n}$. If the cost functions c_1 and c_2 are set to $+\infty$ on all edges outside E, one arrives at the graphic special case of RecovAP for bipartite input graphs G. This allows us to study the problem with graph-theoretic tools, and to look into graph-theoretic structures.
- If the cost function c_1 is set to zero on the edges of some fixed perfect matching and set to $+\infty$ on all the remaining edges, the perfect matching M_1 is thereby fixed and frozen at the zero-cost edges. Then problem RecovAP boils down to finding a matching M_2 that minimizes $c_2(M_2)$ subject to the constraint $|M_1 \cap M_2| \geq k$; we denote the resulting optimization problem as Second-stage recoverable assignment problem (2S-RecovAP).

Both problems RecovAP and 2S-RecovAP are motivated by (central and natural) questions in the area of Recoverable Robust Optimization.

Known and related results. The study of discrete optimization problems with intersection constraints (as imposed in problem RecovAP) was initiated through applications in Recoverable Robust Optimization under interval uncertainty. The literature mainly analyzes situations where the feasible solutions form the bases of various types of matroids: Kasperski & Zieliński [14] construct a polynomial time solution for the case of uniform matroids; the underlying robust optimization problem is called the recoverable selection problem. Lachmann, Lendl & Woeginger [15] provide a simple greedy-type algorithm for recoverable selection, and thereby improve the time complexity in [14] from cubic time down to linear time. Hradovic, Kasperski & Zieliński [12, 11] obtain a polynomial time algorithm for the recoverable matroid basis problem and a strongly polynomial time algorithm for the recoverable spanning tree problem. These results have been generalized and improved by Lendl, Peis & Timmermans [16] who show that the recoverable matroid basis and the recoverable polymatroid basis problem can both be solved in strongly polynomial time. Iwamasa & Takayawa [13] further generalize these results and cover cases with nonlinear and convex cost functions.

Büsing [5] derives various NP-hardness results for recoverable robust shortest s-t-path problems, and thus makes one of the first steps in this area beyond feasible solutions with a matroidal structure. Further results about s-t-paths with intersection constraints are obtained by Fluschnik et al. [8]. Note that the combinatorics of s-t-paths is substantially more complex than the combinatorics of matroid bases: whereas all bases of a matroid have the same cardinality, different s-t-paths may contain totally different numbers of edges. For that reason, recoverable robust shortest s-t-path problems do not (easily) translate into corresponding optimization problems that ask for two feasible solutions with at least k common elements.

Şeref et al. [19] study 2S-RecovAP and obtain a randomized algorithm running in polynomial time if the costs are polynomially bounded. A stable matching variant of 2S-RecovAP has recently been introduced and studied by Bredereck et al. [2].

Our contribution. By analyzing problem RecovAP, we take another step beyond matroidal structures in recoverable robust optimization. Section 2 discusses the computational complexity of RecovAP. We look into the parameterized complexity of RecovAP. We show that

the problem is W[1]-hard with respect to the central parameter k, the lower bound on the intersection size of the two matchings. Furthermore, the problem is W[1]-hard with respect to the so-called recoverability parameter $k' = n - k$, hence the problem that asks to have all of the n matching edges of M_1 and M_2 to coincide except for up to k exceptions. These hardness results even hold in the highly restricted case where both cost functions c_1 and c_2 only take the values 0 and 1. Similar W[1]-hardness results hold for the graphic version of RecovAP on planar graphs. On the positive side, there exists a simple XP algorithm for parameter k (that checks all possible sets $M_1 \cap M_2$ of size k) and there also exists a simple XP algorithm for parameter k' (that checks all possible sets $M_1 - M_2$ and $M_2 - M_1$ of size k'). This is in contrast to the variants of the problem with the constraint $|M_1 \cap M_2| \leq k$ or $|M_1 \cap M_2| = k$. These problems are easily shown to be NP-hard for each fixed k via a reduction from the Disjoint Matchings Problem [9]. Finally, we show that the graphic version of RecovAP with respect to parameter treewidth is in FPT.

Next, in Section 3 we discuss problem RecovAP under Monge-type conditions; we refer to Burkard, Klinz & Rudolf [4] for an extensive overview of Monge properties. The cost function in the assignment problem may naturally be viewed as an $n \times n$ cost matrix. If both cost functions c_1 and c_2 correspond to Monge matrices, problem RecovAP boils down to something trivial: In the optimal solution both matchings M_1 and M_2 run along the main diagonal of the underlying matrix. And if both cost functions c_1 and c_2 correspond to Anti-Monge matrices, then in the optimal solution both matchings M_1 and M_2 run along the secondary diagonal of the underlying matrix. The mixed case where c_1 corresponds to a Monge matrix and where c_2 corresponds to an Anti-Monge matrix is less trivial and more interesting. By analyzing the combinatorial structure of potential optimal solutions, we show that it is solvable in polynomial time.

Finally, in Section 4 we turn to the second-stage recoverable assignment problem 2S-RecovAP, which shows a strange and rather unpleasant behavior. We feel that problem 2S-RecovAP is too hard to allow a polynomial time solution, and we simultaneously feel that it is too easy to allow an NP-hardness proof. We support our intuition by two mathematical arguments: First, by a straightforward reduction to the extact matching problem in red blue biparite graphs by Şeref et al. [19], there exists an RNC^2 algorithm for 2S-RecovAP. As the complexity class $\mathrm{RNC}^2 \subseteq \mathrm{RNC}$ is conjectured to be properly contained in NP, this provides evidence for the easiness of 2S-RecovAP. Secondly, we show that the exact matching problem in red-blue bipartite graphs [7] is logspace reducible to 2S-RecovAP. As the existence of a polynomial time algorithm for this exact red-blue matching problem is doubtful (and constitutes a long-open famous problem), this provides evidence for the hardness of 2S-RecovAP.

Due to the page limit, some of the proofs are omitted or a short sketch is given. The detailed proofs will be published in the full version of the paper.

2 Parameterized Complexity

To show W[1]-hardness of the RecovAP problem we reduce from the well known grid tiling problem. In the grid tiling problem we are given an $\ell \times \ell$ grid in which every cell contains a set of tuples. The task is to select a value for every row and for every column compatible with the tuples in the cells: That is, each cell defined by a row and column combination contains a tuple with the values selected for this row and column.

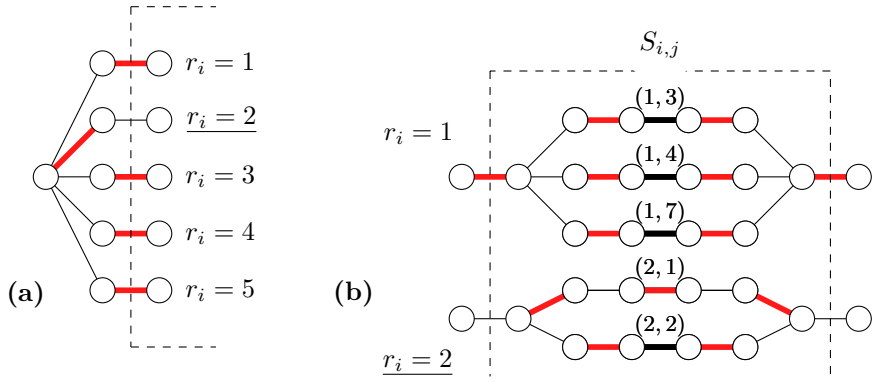


Figure 1 Gadgets for the W[1]-hardness result for RecovAP and parameter k: (a) Selection gadget for row values (analogously for column values). The depicted matching fixes value 2. (b) Component for one row in $S_{i,j}$ (analogously for one column). Each middle edge represents one tuple in $S_{i,j}$ This component exists for both columns and rows and the middle edges are identified if the corresponding tuples are the same.

Grid Tiling

Input: Integers ℓ, n, and a collection $\mathcal{S} = (S_{i,j})_{(i,j)\in[\ell]\times[\ell]}$ with $S_{i,j} \subseteq [n] \times [n]$.

Question: Are there integers $r_1, \ldots, r_\ell$ and $c_1, \ldots, c_\ell$ such that $(r_i, c_j) \in S_{i,j}$ for every $i, j \in [\ell]$?

Grid Tiling has been shown to be $W[1]$-hard for parameter ℓ and has no $f(\ell)n^{o(\ell)}$-time algorithm [6]. For simplicity, we assume that every value $1, \ldots, n$ appears in at least one tuple of $\mathcal{S}$; which can be achieved by renaming the occurring n many values increasingly. That way we ensure that the size of the numbers are polynomial in the size of the input.

By the same reduction, we obtain a lower bound of the runtime assuming the Exponential Time Hypothesis (ETH); for more details on ETH we refer to [6].

▶ **Theorem 1.** *RecovAP is* W[1]*-hard for parameter k with edge cost $(c_1(e), c_2(e)) \in \{(0,0),(0,1),(1,0),(1,1)\}$ for all edges e, and, unless* ETH *fails, it has no $f(k)n^{o(\sqrt{k})}$-time algorithm.*

Proof. Let $\mathcal{I} = (\mathcal{S}, n, \ell)$ be a Grid Tiling instance. We construct a RecovAP instance that asks for matchings M_1 and M_2 of cost 0 and with at least $k = \ell^2$ edges $e \in M_1 \cap M_2$. Hence, only edges with cost $(0,0)$ may be part of $M_1 \cap M_2$. Our construction uses edges with cost $(0,0)$ only to model tuples of grid cells. By our construction, exactly one $(0,0)$ cost edge (r_i, c_j) per grid cell $S_{i,j}$ may be in $M_1 \cap M_2$, which then fixes the selection of that tuple using a cell gadget. We force these tuples to comply with a global selection of row and column values $r_1, \ldots, r_\ell$ and $c_1, \ldots, c_\ell$, using row and column selection gadgets.

In our reduction, we use two types of gadgets, the *row/column selection gadget* and the *cell gadget*. The property of the row/column selection gadget is to encode the selection of one integer per row/column. The role of the cell gadget is to encode the selection of a tuple $(r_i, c_j) \in S_{i,j}$ which is consistent with the selection of the row selection gadget for row i and the column selection gadget for column j. Each of the gadgets will contain a set of special vertices called terminals. These terminals will later be used to connect the gadgets with each other using additional edges.

In the following, we first formally introduce the subgraphs and costs of these gadgets. Next, we combine these gadgets into an instance of RecovAP.

Since we aim for a solution (M_1, M_2) of cost 0, edges of cost 1 are not allowed in the matchings M_1, M_2. Hence, edges with cost $(0,1)$ can only be included in M_1 and edges with cost $(1,0)$ can only be included in M_2. This fact is heavily used in the following arguments.

For the row selection gadget we construct a graph G^{row} which is a star with center vertex v and leaves $t_1, \ldots, t_n$, the terminals of the gadget. All edges of the row selection gadget have cost $(0,1)$. Note, that exactly one of the terminals can be matched with cost 0 using M_1. This matching corresponds to the selected value in the given row. Also, note that in this case the terminal t^r is the unique vertex matched by M_1 and all other terminals must be matched by M_1 to some vertex outside of the gadget. See Figure 1 (a) for an illustration of the row selection gadget, where the gadget is the part left of the dashed box. All the vertices of the row selection gadget will be matched in M_2 using auxiliary vertices introduced at the end of this construction.

Analogously, we construct the column selection gadget as a graph G^{col} with terminals $t^{\text{csel}(c)}$ for all column choices $c \in [n]$. The only difference is, that all edges are assigned cost $(1,0)$ and the selection is determined by the matching M_2.

For the cell gadget we construct a graph $G^{\text{cell}(S)}$, where S is the set of tuples from $[n] \times [n]$ corresponding to the cell. For each tuple $(r,c) \in S$ there exists a special *tuple edge* $e^{\text{tup}(r,c)}$ of cost $(0,0)$ in $G^{\text{cell}(S)}$. For each possible row choice $r \in [n]$ we introduce two terminal vertices $t^{\text{left}(r)}$ and $t^{\text{right}(r)}$. We construct two parts $G^{\text{rcell}(S)}$ and $G^{\text{ccell}(S)}$ handling the row and column selection in the cell (see Figure 1(b) for an illustration of $G^{\text{rcell}(S)}$). $G^{\text{cell}(S)}$ is then defined as the union of $G^{\text{rcell}(S)}$ and $G^{\text{ccell}(S)}$ identifying the common tuple edges.

We begin by introducing the part of the cell gadget that correspond to the row selection, denoted by $G^{\text{rcell}(S)}$. For this part all vertices must be matched by M_1 either inside the gadget or from outside of the gadget if they are terminals. To match these vertices using M_2 we will later introduce auxiliary vertices. The terminals $t^{\text{left}(r)}$ and $t^{\text{right}(r)}$ are part of $G^{\text{rcell}(S)}$ and connected by distinct internally vertex-disjoint paths of length 5 for each tuple $(r,c) \in S$, where $e^{\text{tup}(r,c)}$ is the center edge of this path. All the edges of these paths, except for the tuple edges, are assigned cost $(0,1)$. Note that there might exist r such there is no tuple $(r,c) \in S$. For such r nothing except for the two terminal vertices is added to $G^{\text{rcell}(S)}$. This construction is depicted in Figure 1 (b) inside of the dashed box. Observe that $t^{\text{left}(r)}$ is matched by M_1 from outside of the gadget if and only if $t^{\text{right}(r)}$ is matched by M_1 from outside of the gadget. The only feasible matching in this case adds the second and third edge along the paths from $t^{\text{left}(r)}$ to $t^{\text{right}(r)}$ to M_1. Hence, for such r none of the tuple edges $e^{\text{tup}(r,c)}$ corresonding to a tuple $(r,c) \in S$ can be matched by M_1. But if $t^{\text{left}(r)}$ is not matched by M_1 from outside of the gadget then also $t^{\text{right}(r)}$ cannot be matched from the outside of the gadget and exactly for one tuple $(r,c) \in S$ the tuple edge $e^{\text{tup}(r,c)}$ and the first and last edge along the path connecting $t^{\text{left}(r)}$ to $t^{\text{right}(r)}$ have to be added to M_1. For all other paths the second and third edge have to be added to M_1. This way of selecting tuple edges is illustrated in Figure 1 (b). All the constructed vertices in this paragraph, except those incident to the tuple edges, have to be matched by M_2 using auxiliary vertices.

Analogously, the parts of the cell gadget that correspond to the column selection are denoted by $G^{\text{ccell}(S)}$. For each possible column choice $c \in [n]$ we introduce two terminal vertices $t^{\text{top}(c)}$ and $t^{\text{bottom}(c)}$, analogous to $t^{\text{left}(r)}$ and $t^{\text{right}(r)}$ for the row choices. They are connected to each other via the tuple edges in the same way as the terminals of $G^{\text{rcell}(S)}$. Again by not matching $t^{\text{top}(c)}$ from the outside by M_2 it is enforced that $t^{\text{bottom}(c)}$ is not matched from the outside by M_2 and exactly one tuple edge $e^{\text{tup}(r,c)}$ must be matched by M_2. All the constructed vertices in this paragraph except those incident to the tuple edges will be matched by M_1 using auxiliary vertices.

The cell gadget $G^{\text{cell}(S)}$ is defined as the union of $G^{\text{rcell}(S)}$ and $G^{\text{ccell}(S)}$. The tuple edges and there incident vertices are introduced only once in $G^{\text{cell}(S)}$ and are identified in this union.

In addition we add $2 \cdot (n + |S|)$ auxiliary vertices to $G^{\text{cell}(S)}$. For each $i \in [n]$ we connect $t^{\text{left}(i)}$ with $t^{\text{top}(i)}$ and $t^{\text{right}(i)}$ with $t^{\text{bottom}(i)}$ via a path of length 2, using $2n$ of the auxiliary vertices (see Figure 2 (a)). Observe that for each tuple $(r, c) \in S$ there are 4 vertices connected to the terminals $t^{\text{left}(r)}$, $t^{\text{top}(c)}$, $t^{\text{right}(r)}$ and $t^{\text{bottom}(c)}$ inside $G^{\text{cell}(S)}$. We add a path of length two between the vertex connected to $t^{\text{left}(r)}$ and the vertex connected to $t^{\text{top}(c)}$; and also add a path of length two between the vertex connected to $t^{\text{right}(r)}$ and the vertex connected to $t^{\text{bottom}(c)}$. The cost of the constructed auxiliary edges connected to vertices in $G^{\text{rcell}(S)}$ are set to $(1, 0)$ and the cost of the constructed auxiliary edges connected to vertices in $G^{\text{ccell}(S)}$ are set to $(0, 1)$.

These constructed paths of length 2 can be used to match each of the connected vertices with M_1 or M_2 respectively which also matches the auxiliary vertices with both of the matchings.

Observe that the tuple edges are the only edges in the cell gadget that can both be matched by M_1 and M_2. Hence, if exactly one terminal $t^{\text{left}(r)}$ is not matched by M_1 and exactly one terminal $t^{\text{top}(c)}$ is not matched by M_2 it holds that the cell gadget can contribute one edge to $M_1 \cap M_2$ if and only if $(r, c) \in S$. Using the auxiliary edges all vertices can be matched by both M_1 and M_2.

Now, we are ready to define the instance of RecovAP on a graph G. For each row $i \in [\ell]$ we add two distinct copies of the row gadget $G^{\text{row}}_{i,1} = G^{\text{row}}$, $G^{\text{row}}_{i,2} = G^{\text{row}}$ with terminals $t^{\text{rsel}(r)}_{i,1}, t^{\text{rsel}(r)}_{i,2}$ to G and for each column $j \in [\ell]$ we add two distinct copies of the column selection gadget $G^{\text{col}}_{j,1} = G^{\text{col}}$, $G^{\text{col}}_{j,2} = G^{\text{col}}$ with terminals $t^{\text{csel}(c)}_{j,1}, t^{\text{csel}(c)}_{j,2}$ to G.

For each cell $(i, j) \in [\ell] \times [\ell]$ we add a distinct copy of the cell gadget $G^{\text{cell}}_{i,j} = G^{\text{cell}(S_{i,j})}$ with terminals $t^{\text{left}(r)}_{i,j}$, $t^{\text{right}(r)}_{i,j}$, $t^{\text{top}(c)}_{i,j}$, $t^{\text{bottom}(c)}_{i,j}$ to G.

We now connect the terminals of these gadgets. For each row $i \in [\ell]$ and possible choice r we connect $t^{\text{rsel}(r)}_{i,1}$ to $t^{\text{left}(r)}_{i,1}$ and $t^{\text{rsel}(r)}_{i,2}$ to $t^{\text{right}(r)}_{i,\ell}$ with cost $(0, 1)$. For each column $j \in [\ell]$ and possible choice c we connect $t^{\text{csel}(c)}_{j,1}$ to $t^{\text{top}(c)}_{1,j}$ and $t^{\text{csel}(c)}_{j,2}$ to $t^{\text{bottom}(c)}_{\ell,j}$ with cost $(1, 0)$. For each $i \in [\ell], j \in [\ell - 1]$ and choice r we connect terminals $t^{\text{right}(r)}_{i,j}$ with $t^{\text{left}(r)}_{i,j+1}$ with cost $(0, 1)$. For each $i \in [\ell - 1], j \in [\ell]$ and column choice c we connect terminals $t^{\text{bottom}(c)}_{i,j}$ with $t^{\text{top}(c)}_{i+1,j}$ with cost $(1, 0)$.

To ensure the existence of perfect matchings we add for the vertices of the selection gadgets $2 \cdot (\ell + \ell n)$ additional auxiliary vertices to G. The first half is used to connect the vertices of the row selection gadgets $G^{\text{row}}_{i,1}$ on the left of the grid with the vertices of the column selection gadgets $G^{\text{col}}_{i,1}$ on the top via paths of length 2 for all $i \in [n]$. The others are used to connect the vertices of the row selection gadgets $G^{\text{row}}_{i,2}$ on the right to the column selection gadgets $G^{\text{col}}_{i,2}$ on the bottom via paths of length 2 for all $i \in [n]$. The cost of the edges added in these paths is set to $(1, 0)$ for all edges incident to a vertex of a row selection gadget and $(0, 1)$ for all edges incident to a vertex of a column selection gadget. Hence, all vertices in the row and column selection gadgets can be matched by both matchings with cost 0.

It is now easy to check that the constructed graph is bipartite. To create an instance of RecovAP we add additional edges with cost $(1, 1)$ to obtain a complete bipartite graph. Note that these edges can neither be used by M_1 nor by M_2.

It remains to show that the given instance of the Grid Tiling is a yes-instance if and only if there exists a solution to the constructed instance G of RecovAP with $|M_1 \cap M_2| \geq \ell^2$ of cost 0.

By construction, all the matched tuple edges in each row must be consistent with the row selection and all the matched tuple edges in each column must be consistent with the column selection, else the matchings M_1, M_2 cannot have cost 0. Hence, $|M_1 \cap M_2| \geq \ell^2$ can only be obtained if the same tuple edge is used inside each cell gadget by both the row and column selection.

For the converse direction observe that given a yes-instance of Grid Tiling and the corresponding solution one can easily set the matchings M_1 and M_2 in the constructed gadgets according to the solution for Grid Tiling and obtain a solution for RecovAP of cost 0 such that $|M_1 \cap M_2| \geq \ell^2$.

To show the ETH lower bound assume, for the sake of contradiction, that there is an algorithm for RecovAP with running time $f(k)n^{o(\sqrt{k})}$. Then an instance of Grid Tiling can be transformed in polynomial time into an instance of RecovAP. For the parameter it holds that $k = \ell^2$. So this leads to a running time $f(\ell)n^{o(\sqrt{\ell^2})} = f(\ell)n^{o(\ell)}$. This is a contradiction. ◀

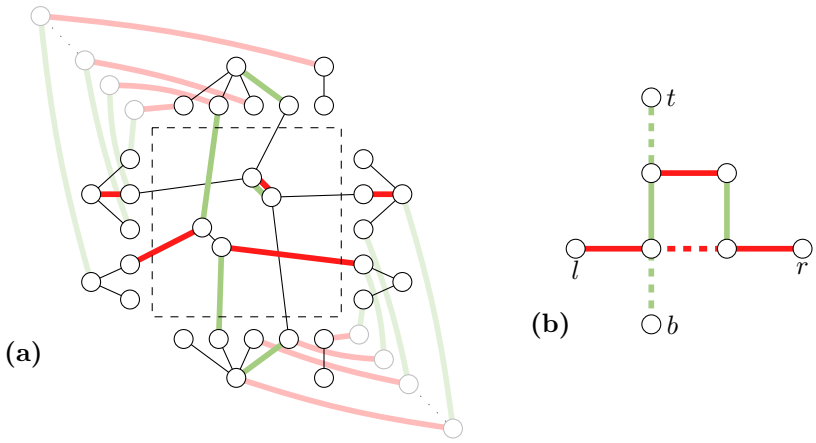


Figure 2 (a) A sketch of the surrounding of a grid cell component (in the dashed box) with matching M_1 in green and M_2 in red. Most of the edges leading into the box are left out, except for two example diagonal $(0,0)$ cost edges, one in both matchings, one in none. Outside in faint color are the additional helper vertices to make the matchings perfect. By positioning these helper vertices on a diagonal as depicted, there are only crossings of edges of cost $(0,1)$ and $(1,0)$. (b) A crossing gadget that replaces a crossing of a $(0,1)$ cost edge $\{l,r\}$ and $(1,0)$ cost edge $\{t,b\}$. The red (dashed and fully drawn) edges have cost $(0,1)$ and the green cost $(1,0)$. The fully drawn red edges show a matching replacing $\{\ell,r\} \in M_1$ while the fully drawn green edges show a matching replacing $\{t,b\} \notin M_2$.

This hardness results also translates to planar graphs with the help of a crossing gadget. Since the input graph is not complete, we no longer need edge cost $(1,1)$.

▶ **Corollary 2.** *RecovAP is* W[1]*-hard on planar graphs for parameter k with $(c_1(e), c_2(e)) \in \{(0,0),(0,1),(1,0)\}$ for all edges e, and unless* ETH *fails it has no $f(k)n^{o(\sqrt{k})}$-time algorithm.*

Proof. The key observation is that by arranging the vertices in the plane as shown in Figure 2(a), there are only crossing edges of cost $(0,1)$ and $(1,0)$. Crossings appear from edges of cost $(0,1)$ in G^{row} and $G^{\text{rcell}(S)}$ with edges of cost $(1,0)$ in G^{col} and G^{ccell}. Also when connecting the auxiliary vertices such crossings appear. Note that when connecting the auxiliary vertices it does not matter which specific vertices are connected. Only the fact that each vertex is connected matters. Hence it is possible to arrange for each cell gadget half of

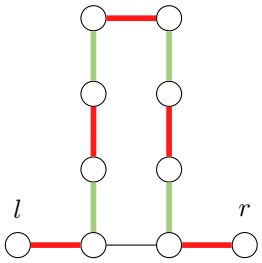


Figure 3 The gadget $G_{l,r}^{\text{snake}(>k,0)}$ that replaces an edge $\{l, r\}$ and effectively simulates an edge of cost $(k + 1, 0)$, for $k = 3$.

the auxiliary vertices in the northwest and half of them in the southeast (see Figure 2(a)). This way it is possible to order the vertices on the left from top to bottom and connect them in this order without any crossings of these $(1, 0)$ cost edges. They then only cross other cost $(0, 1)$ edges of the cell gadget. The principle holds for all other edges connecting the auxiliary vertices of the cell gadget. For the edges connecting auxiliary vertices introduced for the row and column selection gadgets the same idea works by putting them in the northwest and southeast of the whole graph G.

We now introduce the crossing gadget G^{cross} which is used to replace every crossing of an edge $\{\ell, r\}$ of cost $(0, 1)$ with an edge $\{t, b\}$ of cost $(1, 0)$ in G. Graph G^{cross} is illustrated in Figure 2(b). Graph G^{cross} consists of 4 vertices v_1, v_2, v_3, v_4 and the edges $\{\ell, v_1\}$, $\{v_2, r\}$ and $\{v_3, v_4\}$ of cost $(0, 1)$ and the edges $\{t, v_3\}$, $\{v_3, v_1\}$ and $\{v_4, v_2\}$ of cost $(1, 0)$.

Observe that the case $\{\ell, r\} \in M_1$ is simulated by $\{\ell, v_1\} \in M_1$, $\{v_2, r\} \notin M_1$ and $\{v_3, v_4\} \in M_1$ and the case $\{\ell, r\} \notin M_1$ is simulated by $\{\ell, v_1\} \notin M_1$, $\{v_2, r\} \in M_1$ and $\{v_3, v_4\} \notin M_1$. Similarly, there are two ways to simulate $\{t, b\} \in M_2$ and $\{t, b\} \notin M_2$.

It is important that the four new vertices are always matched within this crossing gadget, and thus no further auxiliary vertices are needed. Further, note that this construction can be easily chained in order to handle cases where $\{\ell, r\}$ or $\{t, b\}$ cross more than one other edge. ◀

As a final step regarding planar graphs, we avoid vertices of degree > 4, and we avoid $(0, 0)$ cost edges, thus showing hardness for cost $(0, 1)$ and $(1, 0)$. The key step is to replace a single $(0, 1)$ edge by a long path-like gadget that effectively simulates a cost $(0, k + 1)$ edge, analogously for a $(1, 0)$ edge. Figure 3 shows such a gadget.

▶ **Corollary 3.** *RecovAP is* W[1]*-hard on planar graphs with maximum degree 4 for parameter k with $(c_1(e), c_2(e)) \in \{(0, 1), (1, 0)\}$ for all edges e, and unless* ETH *fails it has no $f(k)n^{o(\sqrt{k})}$-time algorithm.*

Using similar ideas we can also show W[1]-hardness for the dual parameter $k' = n - k$, hence the problem that asks to have all of the n matching edges of M_1 and M_2 to coincide except for up to k exceptions. In robust optimization this parameter is of importance and called the recoverability parameter.

▶ **Theorem 4.** *RecovAP is* W[1]*-hard for the recoverability parameter $k' = n - k$ with $(c_1(e), c_2(e)) \in \{(0, 0), (0, 1), (1, 0), (1, 1)\}$ for all edges e, and, unless* ETH *fails, it has no $f(k)n^{o(\sqrt{k})}$-time algorithm.*

Ideally, Theorem 4 would translate to planar graphs by using a crossing gadget. However, according to Gurjar et al. [10], such a crossing gadget does not exist in this case.

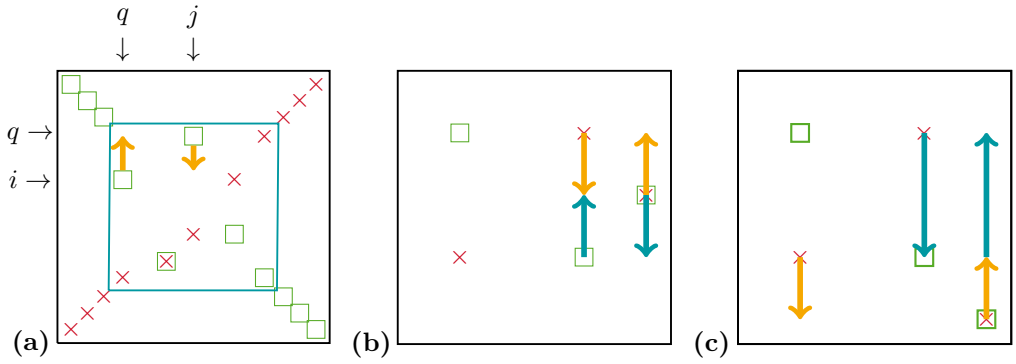


Figure 4 Illustrations for the Monge and Anti-Monge case. (a) Modification to ensure that no cycles have length larger than four. (b) Modification to move 2-cycles east of a 4-cycle into a 4-cycle. (c) Modification to move 2-cycles southeast of a 4-cycle into a 4-cycle.

Beside planar graphs, we also consider graphs of bounded treewidth. RecovAP is fixed parameter tractable in the treewidth of the input graph (without the intersection size k as parameter). Our algorithm is based on dynamic programming over the tree decomposition.

▶ **Theorem 5.** *RecovAP is in* FPT *with respect to the treewidth of the input graph.*

3 Monge and Anti-Monge Matrices

In this section we develop a polynomial time algorithm for the special case of RecovAP if the cost function c_1 is given by a Monge matrix $A = (a_{i,j}) \in \mathbb{R}^{n\times n}$ and the cost function of c_2 is given by an Anti-Monge matrix $B = (b_{i,j}) \in \mathbb{R}^{n\times n}$. The matrix A is called a Monge matrix if for all $i < k$ and $j < l$ it holds that $a_{i,j} + a_{k,l} \leq a_{i,l} + a_{k,j}$. Analogously, the matrix B is called an Anti-Monge matrix if for all $i < k$ and $j < l$ it holds that $b_{i,j} + b_{k,l} \geq b_{i,l} + b_{k,j}$. Let $U = \{u_1, \ldots, u_n\}$ and $V = \{v_1, \ldots, v_n\}$ be the bipartition of the vertex set of $K_{n,n}$. Then the costs of edge $\{u_i, v_j\}$ is given by $c_1(\{u_i, v_j\}) = a_{i,j}$ and $c_2(\{u_i, v_j\}) = b_{i,j}$.

Note, that it is a well-known result that if the cost function of the AP is given by a Monge matrix then the diagonal $\{\{u_i, v_i\} \colon i = 1, \ldots, n\}$ is an optimal solution, similarly the anti-diagonal for Anti-Monge matrices.

Surprisingly, for these special cost functions also the RecovAP has an optimal solution of a similar combinatorial structure, as we show in the following. We illustrate solutions in matrix form by highlighting entry (i, j) with a square if $\{u_i, v_j\} \in M_1$ and with a cross if $\{u_i, v_j\} \in M_2$.

▶ **Theorem 6.** *Let c_1 be given by a Monge matrix A and c_2 be given by an Anti-Monge matrix B. Then, there exists an optimal solution M_1, M_2 to the RecovAP such that*

1. *$\{u_i, v_i\} \in M_1$ for all $i = 1, \ldots, \lfloor \frac{n-k}{2} \rfloor$ and $i = \lceil \frac{n+k}{2} \rceil, \ldots, n$,*
2. *$\{u_i, v_{n+1-i}\} \in M_2$ for all $i = 1, \ldots, \lfloor \frac{n-k}{2} \rfloor$ and $i = \lceil \frac{n+k}{2} \rceil, \ldots, n$,*
3. *$M_1 \cap M_2$ is an optimal solution to the AP with cost $c_1 + c_2$ on the complete bipartite subgraph induced by the sets of vertices $\{u_i \mid i = \lfloor \frac{n-k}{2} \rfloor + 1, \ldots, \lceil \frac{n+k}{2} \rceil - 1\}$ and $\{v_i \mid i = \lfloor \frac{n-k}{2} \rfloor + 1, \ldots, \lceil \frac{n+k}{2} \rceil - 1\}$.*

Based on this structural result we can easily compute an optimal solution for RecovAP by solving the instance of the AP on the subgraph stated in point 3 of Theorem 6 and then completing the perfect matchings M_1 and M_2 as stated in points 1 and 2. In summary, we obtain the following result.

▶ **Theorem 7.** *Let c_1 be given by a Monge matrix A and c_2 be given by an Anti-Monge matrix B. Then the* RecovAP *can be solved in $O(n + k \log k)$ time.*

In the following we prepare the proof of Theorem 6 based on several structural lemmas. The main tool to analyze feasible solutions M_1, M_2 is their decomposition into M_1-M_2-alternating cycles of even length. For any even number s we call such an alternating cycle an s-cycle. Using this language, we call an edge $e \in M_1 \cap M_2$ a 2-cycle. A 4-cycle consists of four edges $\{u_i, v_j\}, \{u_{i'}, v_{j'}\} \in M_1$ and $\{u_i, v_{j'}\}, \{u_{j'}, v_i\} \in M_2$. Note that by the fact that A is Monge and B is Anti-Monge the cost of such edges is minimum if $i < i'$ and $j < j'$. We call a 4-cycle fulfilling this property an aligned 4-cycle. We identify the 4-cycle with its indices (i, j, i', j'). Similarly, we identify a 2-cycle $\{v_i, v_j\} \in M_1 \cap M_2$ with its indices (i, j). Also, observe that in our matrix visualization 4-cycles correspond to 2×2 submatrices where the corners diagonal to each other are marked by squares and crosses. The fact that $i < i'$ and $j < j'$ implies that the squares are drawn into the northwest and southeast corner and the crosses are drawn into the northeast and southwest corners. We say that a 4-cycle (i_1, j_1, i'_1, j'_1) is nested inside another 4-cycle (i_2, j_2, i'_2, j'_2) if it holds that $i_2 < i_1 < i'_1 < i'_2$ and $j_2 < j_1 < j'_1 < j'_2$. Analogously we say that a 2-cycle (i_1, j_1) is nested inside a 4-cycle (i_2, j_2, i'_2, j'_2) if $i_2 < i_1 < i'_2$ and $j_2 < j_1 < j'_2$.

Note, that in the language of such cycles Theorem 6 is equivalent to: there exists an optimal solution which consists of $\lfloor \frac{n-k}{2} \rfloor$ many aligned 4-cycles and all the other matching edges are 2-cycles; all 4-cycles are nested into each other and the 2-cycles are nested inside the innermost 4-cycle; the 2-cycles form a minimum cost perfect matching with respect to $c_1 + c_2$ on their vertices.

In Lemma 8 we prove that there always exists an optimal solution without s-cycles for $s > 4$, and all the 4-cycles are nested and aligned. The main idea here is to iteratively remove such long cycles from the outside to the inside. In a second step (Lemma 9) we then show that there exists an optimal solution in which all 2-cycles lie inside the innermost 4-cycle.

▶ **Lemma 8.** *Let perfect matchings M_1, M_2 be feasible solutions to 2S-*RecovAP*. Then there exists a solution M'_1, M'_2 consisting only of 2-cycles and aligned 4-cycles, and all the 4-cycles are nested.*

Proof. As a first step consider the subinstance (submatrices) where all vertices (rows and columns) contained in 2-cycles are removed (which makes handling row an column indices easier in the following). The resulting submatrices of A and B remain Monge and Anti-Monge.

Now assume that for $q = 1, \dots, \ell - 1$ it we have that $(q, q, n+1-q, n+1-q)$ already forms a 4-cycle in M_1, M_2. Note that as base of the induction the case $\ell = 1$ trivially true. We now construct matchings M'_1, M'_2 of smaller or equal cost such that $(q, q, n + 1 - q, n + 1 - q)$ is also a 4-cycle for $q = \ell$. If $\{u_q, v_q\} \notin M_1$ it there are edges $\{u_i, v_q\}, \{u_q, v_j\} \in M_1$. We have that $q < i < n + 1 - q$ and $q < j < n + 1 - q$ due to our assumption on present 4-cycles. Because A is Monge, we can exchange those edges for the edges $\{u_q, v_q\}, \{u_i, v_j\}$ in M'_1. See Figure 4 (a) for an illustration of this modification. Analogously, we can ensure that M'_1 also contains $\{u_{n+1-q}, v_{n+1-l}\}$ and M'_2 contains both $\{u_{n+1-q}, v_q\}, \{u_q, v_{n+1-q}\}$, forming the 4-cycle as claimed.

By induction we obtain a solution consisting of only nested aligned 4-cycles, except for maybe one additional 2-cycle exactly in the center of the matrix. We obtain the solution M'_1, M'_2 as claimed by adding back the vertices (rows and columns) of the 2-cycles removed in the first step. ◀

▶ **Lemma 9.** *There is a solution M_1, M_2 with minimum cost $c_1(M_1) + c_2(M_2)$ where all 4-cycles are nested and aligned, and where no 2-cycle is outside of a 4-cycle.*

Proof. Note that the first part of the claim is already implied by Lemma 8, and we start with matchings M_1, M_2 fulfilling the structure stated in Lemma 8. For the second claim we again process the cycles from outside to inside with respect to the nesting order of the 4-cycles. Assume that (x, y) is the outmost 2-cycle in M_1, M_2, and let (i, j, i', j') be the outmost 4-cycle that does not contain (x, y). We show how to modify M_1, M_2 such that the cost does not increase, the number of 2-cycles does not decrease and such that the number of 4-cycles that contain all 2-cycles is increased by one.

We do this by looking at two distinct cases (up to symmetry). Case 1, we have $i < j < x$ and $i' < y < j'$, i.e., (x, y) lies east of (i, j, i', j'). In this case we remove $\{u_i, v_{j'}\}$ and $\{u_x, u_y\}$ from M_2 and add $\{u_x, v_{j'}\}$ and $\{u_i, v_y\}$ to M_2, which can only improve the cost, since B is Anti-Monge. In addition we remove $\{u_{i'}, v_{j'}\}$ and $\{u_x, v_y\}$ from M_1 and add $\{u_x, v_{j'}\}$ and $\{u_{i'}, v_y\}$ to M_1. See Figure 4 (b) for this modification. Note, that now (i, j, i', y) is a new 4-cycle containing the new 2-cycle (x, j'). A 2-cycle lying to the north, south or west of the 4-cycle can be handled symmetrically.

Case 2 is the case when the 2-cycle lies to the southeast of the 4-cycle, i.e. $i < i' < x$ and $j < j' < y$. In this case we remove $\{u_{i'}, v_j\}$ and $\{u_x, v_y\}$ from M_2 and replace it with $\{u_x, v_j\}$ and $\{u_{i'}, v_y\}$ which can only decrease the cost by the fact that B is Anti-Monge. As a second step we remove $\{u_i, v_{j'}\}$ and $\{u_{i'}, v_y\}$ from M_2 and add $\{u_{i'}, v_{i'}\}$ and $\{u_i, v_y\}$ to M_2. See Figure 4 (c) for this modification. Note, that now (i, j, x, y) is a new 4-cycle containing the new 2-cycle (i', j'). The cases when the 2-cycle lies northeast, southwest or northwest can be handled similarly. ◀

Now we are ready to give the proof of Theorem 6.

Proof of Theorem 6. Basically Lemma 9 already implies the combinatorial structure claimed in Theorem 6. The only point missing is that there are exactly $\lfloor \frac{n-k}{2} \rfloor$ many nested 4-cycles in the solution and the remaining edges form 2-cycles inside.

Note that selecting more 2-cycles than strictly necessary (by the constraint or the combinatorial structure) is never helpful, since because of the Monge structure 4-cycles correspond to the optimal solution of the two independent APs.

Hence, if 2 divides $n - k$ the theorem follows directly, there is an optimal solution with k 2-cycles at positions i, j with $\frac{n-k}{2} < i, j < \frac{n-k}{2}$.

Otherwise, if $n - k$ is not multiple of 2, we can assume that n is even and k is odd, since if n is odd an optimal solution always selects the edge $\{u_{(n+1)/2}, v_{(n+1)/2}\}$ as a 2-cycle, giving an equivalent instance with $n - 1$ rows and columns and the constraint to select at least $k - 1$ many 2-cycles. Hence let n be even and k odd. Since the number of edges in a 2-cycle is even, we must select at least $k + 1$ many 2-cycles, and the claim follows. ◀

4 The Second Stage Recoverable Assignment Problem

In this section we study a variant of RecovAP in which the perfect matching M_1 is fixed and we are looking for a perfect matching M_2 of minimum linear cost $c_2(M_2)$ subject to the constraint that $|M_1 \cap M_2| \geq k$. Note that 2S-RecovAP is a special case of RecovAP, with the special cost structure $c_1(e) = 0$ if $e \in M_1$ and $c_1(e) = \infty$ otherwise.

Using the language of recoverable robust optimization this problem is called the incremental assignment problem. Şeref et al. [19] study this problem and obtain a straightforward reduction to Exact Matching in Red-Blue Bipartite Graphs, one of the few natural problems known to be in Randomized-NC (RNC) for which no polynomial time algorithm is known.

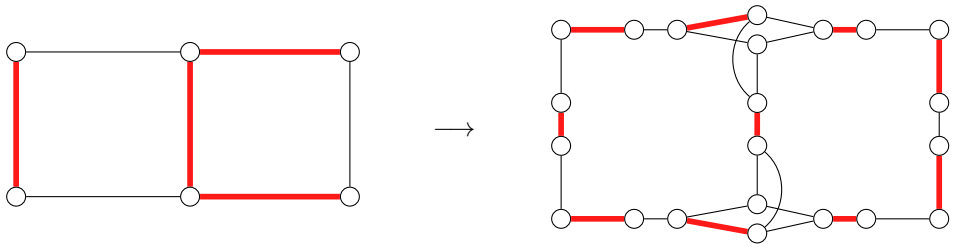


Figure 5 Visualization of the reduction from the special case of Exact Matching in Red-Blue Bipartite Graphs where R is a matching to general Exact Matching in Red-Blue Bipartite Graphs.

Exact Matching in Red-Blue Bipartite Graphs

Input: A bipartite graph $G = (U \cup V, E)$, a subset of edges $R \subseteq E$ (the red colored edges), and an integer $k \in \mathbb{N}$.

Question: Is there a perfect matching M of G such that $|M \cap R| = k$.

Mulmuley et al. [17] show that this problem can be solved in randomized polynomial time if all costs are polynomially bounded. In summary, the following result holds.[2]

▶ **Corollary 10** (Şeref et al. [19]). *2S-RecovAP can be solved by an* RNC^2 *algorithm, if all costs c_2 are polynomially bounded.*

The techniques for this algorithm are not specific to bipartite graphs. Hence they can also be used to solve the second stage of the recoverable perfect matching problem on general graphs in RNC^2.

Surprisingly, we are able to prove that the complexity of these problems is essentially equal. We show that 2S-RecovAP is at least as hard as Exact Matching in red-blue bipartite graphs. Note here that our following logspace reduction implies a reduction in NC^2 [18].

▶ **Theorem 11.** *Exact Matching in Red-Blue Bipartite Graphs is logspace reducible to 2S-RecovAP.*

Proof. As a first part of the proof we give a reduction from exact matching in red-blue colored bipartite graphs to the special case of the problem where the set of red edges forms a matching. In the second part we then show that we can reduce this problem to 2S-RecovAP.

For this first part let $((G = (U \cup V), E), R, k)$ be the given instance of exact matching. We construct a new bipartite graph $G'' = (U'' \cup V'', E'')$ and a set of edges $R'' \subseteq E''$ (see Figure 5 for an illustration). Note that without loss of generality we can assume that G contains no vertex of degree one, since such vertices can always be preprocessed in a trivial way.

For every vertex v in G we add an independent set $v_1, \ldots, v_{\deg(v)-1}$ of $\deg(v) - 1$ many vertices to G''. If $v \in U$ the vertices are added to U'', otherwise if $v \in V$ the vertices are added to V''. For every edge $e = \{u, v\}$ of G we add two vertices u_e and v_e to G'' and connect them by the edge $\{u_e, v_e\}$. If $\{u, v\} \in R$. Then we add $\{u_e, v_e\}$ to R''. In addition we add all the edges $\{u_i, u_e\}$ for $i = 1, \ldots, \deg(u) - 1$ and $\{v_i, v_e\}$ for $i = 1, \ldots, \deg(v) - 1$ to E''. Observe that the graph G'' is bipartite if and only if G is bipartite and note that since $|U| = |V|$ also $|U''| = |V''|$.

[2] For formal definitions of NC^2 and RNC^2 see [18, Sections 15.3 and 15.4].

▷ Claim (a). There exists a perfect matching of G with exactly k edges in R if and only if there exists a perfect matching in G'' with exactly k edges in R''.

Proof. Given a perfect matching M of G with exactly k edges in R we construct a perfect matching M'' in G'' with exactly k edges in R''. For each $e \in M$ we add the edge $\{v_e, u_e\}$ to M''. Note that this way we add exactly k edges from R'' to M''. Now for every vertex v exactly one of its incident edges in G is in M. Hence there are exactly $\deg(v) - 1$ incident edges that are not in M. For each such edge $\{v, w\} \in E \setminus M$ we select one of the vertices v_j for $j \in \{1, \ldots, \deg(v) - 1\}$ and add $\{v_j, v_e\}$ to M''. Note that this way M'' is a perfect matching in G'' with exactly k edges from R'' in M''.

For the converse direction, assume that M'' is a perfect matching of G'' with exactly k edges from R''. We construct a perfect matching M of G with exactly k edges from R. Note that every original vertex $v \in V$ is replaced by an independent set $v_1, \ldots, v_{\deg(v)-1}$ in G''. Each of the vertices $v_1, \ldots, v_{\deg(v)-1}$ is matched to a vertex $v_{e'}$ for an incident edge $e' \in E$. But since there exist only $\deg(v) - 1$ such vertices and v has exactly $\deg(v)$ incident edges in G there exists a unique edge $e = \{v, w\}$ in G for which v_e is not matched to one of $v_1, \ldots, v_{\deg(v)-1}$. Hence v_e must be matched to its only remaining neighbor w_e by M''. Note, that by similar arguments e is also the unique incident edge $e' = \{v, w\}$ in G for which the vertex $w_{e'}$ is not matched to one of the vertices $w_1, \ldots, w_{\deg(w)-1}$. We add the edge $\{v, w\}$ to M. Since v is an arbitrary vertex in G such an edge is added for every v, hence M is a perfect matching in G. Since M'' contains exactly k edges from R'' and for each edge $\{v_e, w_e\}$ in M'' we add the edge $e = \{v, w\}$ to M, also M contains exactly k edges from R. ◁

As a next step we show how to obtain the instance (G', M_1, c_2, k) of 2S-RecovAP such that there exists a perfect matching of G with exactly k edges in R if and only if the optimal value for $(G' = (U' \cup V', E'), M_2, c_2, k)$ is k. The graph G'' constructed above is a subgraph of G'. In addition to that, for each vertex $v \in V''$ that is not matched by R'' we add an additional vertex v' to G' and the edge $\{v, v'\}$. Since G'' is a bipartite graph with $|U''| = |V''|$ and R'' is a matching we can select for each such v' another unique vertex u' and add the edge $\{v', u'\}$ to G'. We define the set R' as the set of edges consisting of R'' and all edges $\{v, v'\}$ for all $v \in \{u \in V'' \colon u \text{ not matched by } R''\}$. Note that R' is a perfect matching in G'. We set $c_2(e) = 1$ for all $e \in R''$ and $c_2(e) = \infty$ for all edges $\{v, v'\}$ where $v \in \{u \in V'' \colon u \text{ not matched by } R''\}$. For all other edges the cost c_2 is equal to 0.

▷ Claim (b). There exists a perfect matching of G with exactly k edges in R if and only if there exists solution to 2S-RecovAP instance (G', M_1, c_2, k) with cost k.

Proof. Assume that there exists a perfect matching M of G with exactly k edges in R. Then by Claim (a) there also exists a perfect matching M'' of G'' with exactly k edges in G''. Based on M'' we define a perfect matching M_2 in G' in the following way. The matching M'' is added to M_2 and hence all vertices in the subgraph G'' of G' are matched. For all the remaining vertices, by the construction above there exists a unique matching consisting of the edges $\{v', u'\}$ which are added to M_2. Note, that $c_2(M_2) = k$.

For the converse direction, assume that M_2 is a perfect matching in G'' with at least k edges from M_1 and cost k. Since only edges in $R' \cap M_1$ have finite cost and $c_2(M_2) = k$ it holds that exactly k edges from R' are contained in M_2. In addition, since none of the edges $\{v, v'\}$ are contained in M_2 it holds that $M_2 \cap E'$ is a perfect matching in G' with exactly k edges from R'. Hence by Claim (a) there exists a perfect matching in G containing exactly k edges from R. ◁

This completes the reduction as claimed in the theorem. Note that this reduction can be implemented using logarithmic space. We just have to process one vertex after another and need to implement a counter counting up to the degree of a vertex. ◀

The case with costs c_2 that are not polynomially bounded remains open. But note, that an RNC algorithm for this problem would imply an RNC algorithm for the special case of obtaining a minimum cost perfect matching, which is a long standing open problem [7].

References

1 Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

2 Robert Bredereck, Jiehua Chen, Dušan Knop, Junjie Luo, and Rolf Niedermeier. Adapting stable matchings to evolving preferences. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1830–1837, 2020.

3 Rainer Burkard, Mauro Dell'Amico, and Silvano Martello. *Assignment Problems: Revised Reprint*. SIAM, 2012.

4 Rainer E. Burkard, Bettina Klinz, and Rüdiger Rudolf. Perspectives of Monge properties in optimization. *Discret. Appl. Math.*, 70(2):95–161, 1996. doi:10.1016/0166-218X(95)00103-X.

5 Christina Büsing. Recoverable robust shortest path problems. *Networks*, 59(1):181–189, 2012.

6 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. Parameterized algorithms, 2015. URL: http://ebooks.ciando.com/book/index.cfm/bok_id/1960687.

7 Egres. Exact matching in red-blue bipartite graphs. http://lemon.cs.elte.hu/egres/open/Exact_matching_in_red-blue_bipartite_graphs. Accessed: 2020-07-13.

8 Till Fluschnik, Rolf Niedermeier, Carsten Schubert, and Philipp Zschoche. Multistage st path: Confronting similarity with dissimilarity in temporal graphs. In *31st International Symposium on Algorithms and Computation (ISAAC 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

9 Alan M Frieze. Complexity of a 3-dimensional assignment problem. *European Journal of Operational Research*, 13(2):161–164, 1983.

10 Rohit Gurjar, Arpita Korwar, Jochen Messner, Simon Straub, and Thomas Thierauf. Planarizing gadgets for perfect matching do not exist. In *International Symposium on Mathematical Foundations of Computer Science*, pages 478–490. Springer, 2012.

11 Mikita Hradovich, Adam Kasperski, and Paweł Zieliński. Recoverable robust spanning tree problem under interval uncertainty representations. *Journal of Combinatorial Optimization*, 34(2):554–573, 2017.

12 Mikita Hradovich, Adam Kasperski, and Paweł Zieliński. The recoverable robust spanning tree problem with interval costs is polynomially solvable. *Optimization Letters*, 11(1):17–30, 2017.

13 Yuni Iwamas and Kenjiro Takayawa. Optimal matroid bases with intersection constraints: Valuated matroids, m-convex functions, and their applications. In *Proceedings of the 16th Annual Conference on Theory and Applications of Models of Computation (TAMC 2020), to appear*, 2020.

14 Adam Kasperski and Paweł Zieliński. Robust recoverable and two-stage selection problems. *Discrete Applied Mathematics*, 233:52–64, 2017.

15 Thomas Lachmann, Stefan Lendl, and Gerhard J. Woeginger. A linear time algorithm for the robust recoverable selection problem. *Discret. Appl. Math.*, 303:94–107, 2021. doi:10.1016/j.dam.2020.08.012.

16 Stefan Lendl, Britta Peis, and Veerle Timmermans. Matroid bases with cardinality constraints on the intersection. *arXiv preprint arXiv:1907.04741*, 2019.

17 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987. doi:10.1007/BF02579206.

18 Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

19 Onur Şeref, Ravindra K. Ahuja, and James B. Orlin. Incremental network optimization: Theory and algorithms. *Operations Research*, 57(3):586–594, 2009. doi:10.1287/opre.1080.0607.

Dynamic Data Structures for Timed Automata Acceptance

Alejandro Grez
Pontificia Universidad Católica de Chile, Santiago, Chile
Millennium Institute for Foundational Research on Data, Santiago, Chile

Filip Mazowiecki
Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany

Michał Pilipczuk
University of Warsaw, Poland

Gabriele Puppis
University of Udine, Italy

Cristian Riveros
Pontificia Universidad Católica de Chile, Santiago, Chile
Millennium Institute for Foundational Research on Data, Santiago, Chile

Abstract

We study a variant of the classical membership problem in automata theory, which consists of deciding whether a given input word is accepted by a given automaton. We do so through the lenses of parameterized dynamic data structures: we assume that the automaton is fixed and its size is the parameter, while the input word is revealed as in a stream, one symbol at a time following the natural order on positions. The goal is to design a dynamic data structure that can be efficiently updated upon revealing the next symbol, while maintaining the answer to the query on whether the word consisting of symbols revealed so far is accepted by the automaton. We provide complexity bounds for this dynamic acceptance problem for timed automata that process symbols interleaved with time spans. The main contribution is a dynamic data structure that maintains acceptance of a fixed one-clock timed automaton $\mathcal{A}$ with amortized update time $2^{\mathcal{O}(|\mathcal{A}|)}$ per input symbol.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases timed automata, data stream, dynamic data structure

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.20

Related Version *Full Version*: https://arxiv.org/abs/2002.07049

Funding *Alejandro Grez*: This work was supported by ANID – Millennium Science Initiative Program – Code ICN17_002.
Michał Pilipczuk: This work is a part of project TOTAL that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, grant agreement No. 677651.
Gabriele Puppis: This work was partly supported by ANR project DELTA, grant ANR-16-CE40-0007.
Cristian Riveros: This work was supported by ANID – Millennium Science Initiative Program – Code ICN17_002.

1 Introduction

Imagine we would like to monitor whether the behavior of a server is correct. The run of the server can be abstracted by an infinite stream $w = a_1 a_2 a_3 \ldots \in \Sigma^\omega$, where Σ is a finite alphabet of possible events. The events are disclosed one at a time on the input, and at every moment we should tell whether the prefix consisting of the events observed so far is correct.

A simple yet expressive formalism for describing properties of such *data streams* is provided by classical finite automata. For example, suppose we would like to verify the property that a certain resource is being used by at most one process. Assume that the alphabet is $\Sigma = \{o, r\} \cup \Gamma$, where o denotes a request of the resource, r denotes a release of the resource, and Γ contains other immaterial events. The streams satisfying the discussed property can be then characterized as those where every prefix is accepted by the two-state automaton $\mathcal{A}$ of Figure 1. Here, a state indicates whether the resource is currently available or not.

Verifying the correctness of a stream over time can be formalized through the following *dynamic acceptance problem*: for a fixed automaton $\mathcal{A}$, design a *data structure* that upon receiving subsequent events from the stream, monitors whether the prefix read so far is accepted by $\mathcal{A}$. An obvious, though usually suboptimal solution would be to store in the data structure the prefix read so far, and, upon receiving a new symbol, run the automaton on the whole prefix. This would require time linear in the total length of the prefix, which after a while can become very large compared to $|\mathcal{A}|$, the size of the automaton $\mathcal{A}$. So we would like to minimize the update time by smartly organizing and reusing information computed before.

Cast in this way, the dynamic acceptance problem naturally lends itself to a treatment using the notions of parameterized complexity. Namely, we consider the automaton $\mathcal{A}$ fixed and use the parameter $|\mathcal{A}|$ as an auxiliary measure for expressing guarantees on the update time. Ideally, we would like to obtain update time bounded by a computable function of $|\mathcal{A}|$ only. This way, our work inscribes into the area of *parameterized dynamic data structures*, which is a direction that is still relatively unexplored, but starts to attract considerable attention; see e.g. [3, 7, 11] and references therein for an overview of recent advances.

For finite automata, the dynamic acceptance problem can be solved easily with update time $\mathcal{O}(|\mathcal{A}|)$, as follows. After reading a prefix u, the data structure stores the subset of states $S \subseteq Q$ in which the automaton may be after reading u (in general, we allow the automaton to be non-deterministic). Upon receiving the next input symbol, the set S is updated by applying the possible transitions on every state in S. Moreover, telling whether $\mathcal{A}$ accepts the current input prefix boils down to checking whether S contains an accepting state. Both the update and the query described above can be implemented in time linear in $|\mathcal{A}|$.

Unfortunately, real-life scenarios involve many aspects that cannot be captured by a simple formalism such as finite automata. One of these aspects is *time*. Consider the following example of property that needs to be verified: at every moment in time when an event occurs, a backup operation has been performed within the last 24 hours. A natural choice to model this and similar properties is to enhance finite automata with the ability of measuring time, by adding one or more *clocks*. A definition of the resulting automaton model, called *timed automaton*, is presented in Section 2. Intuitively, a possible timed automaton for the considered property would have one clock $\mathtt{x}$ and two states, "before backup" and "after backup", and would behave as follows (see the right hand-side of Figure 1). The idea is that while processing an input prefix u, the automaton non-deterministically guesses a single backup event b and verifies that this event occurred within the last 24 hours. Thus, upon reading an occurrence of event b, the automaton may either ignore this event and carry on,

Figure 1 Left: a finite automaton $\mathcal{A}$ recognising language $\Gamma^*(o\Gamma^* r\Gamma^*)^*(\{\varepsilon\} \cup o\Gamma^*)$, where occurrences of o are interleaved by occurrences of r. Right: a timed automaton $\mathcal{B}$ with single clock $\mathtt{x}$.

or move from state "before backup" to state "after backup" and reset the clock. The input prefix u is accepted if the automaton reached state "after backup" and, during events since the last reset, the value of the clock has never exceeded 24 hours.

Timed automata are a central topic in the area of verification, and they have a rich and diverse literature, see e.g. [4, 8, 12]. In this work we are interested in the dynamic acceptance problem for timed automata, defined analogously to that for finite automata.

Note that in the setting of timed automata, the same technique that worked for finite automata will not work so easily. The reason is that for a finite automaton $\mathcal{A}$, the set of configurations in which $\mathcal{A}$ may be is a subset of the set of control states, whose size is bounded by the size of $\mathcal{A}$. On the other hand, a configuration of a timed automaton consists of a control state and a tuple of clock values, so the number of possible configurations is a priori unbounded. Concretely, after reading a prefix of length n, there may be as many as $\mathcal{O}(|\mathcal{A}| \cdot n^k)$ different configurations which the given k-clock timed automaton may possibly reach, due to non-determinism and clock resets. Efficient maintenance of this configuration set in a data structure poses the main conceptual challenge in this paper.

Our contribution. We design a dynamic data structure that, for a fixed timed automaton $\mathcal{A}$ with *one* clock, monitors whether $\mathcal{A}$ accepts the prefix read so far with amortized update time $2^{\mathcal{O}(|\mathcal{A}|)}$. This can be improved to worst-case (i.e. non-amortized) update time when the input stream is *discrete*, that is, when all time spans between consecutive events are equal. Our data structure actually works in a slightly more general setting, where the automaton $\mathcal{A}$ is not entirely fixed, but rather is provided on input upon initialization of the data structure.

We also give a somewhat complementary lower bound: under the 3SUM Conjecture, we prove that there exists a fixed timed automaton $\mathcal{A}$ with two clocks and additive constraints on them such that no data structure for the dynamic acceptance problem for $\mathcal{A}$ may achieve strongly sublinear amortized update time (i.e. time $\mathcal{O}(n^{1-\delta})$ for $\delta > 0$). Here, by additive constraints we mean that in the transition relation of $\mathcal{A}$ we may use affine clock conditions that involve more than one clock, e.g. $\mathtt{x} + \mathtt{y} = c$ where $\mathtt{x}, \mathtt{y}$ are clocks and c is a constant.

If the given timed automaton $\mathcal{A}$ has more than one clock, but only constraints involving a single clock are allowed, it remains open whether there is an efficient data structure for the dynamic acceptance problem or a lower bound similar to the above one.

Related work. The setting in this work is close to *runtime verification* [20], an area that focuses on verification techniques that could be performed at runtime, e.g. using timed automata [26, 10]. However, while we study monitoring a data stream through a suitable data structure in the *dynamic* setting, studies on runtime verification typically focus on *static* problems. An example of such a problem is: given an input prefix u, verify whether there is a sequence of events that extends u to a word accepted by the device (e.g. a finite automaton). The problem studied in [25] is similar to the setting presented here; however, this line of work considers constants (e.g. 24 in Figure 1) as part of the input contributing to the considered parameter, and this considerably simplifies the problem (see Section 2 and 3).

The dynamic acceptance problem that we consider here resembles the setting of *streaming algorithms*; see e.g. [5, 13, 17] for works with a similar motivation. In this context, a typical problem is to compute (possibly approximately) some statistics or an aggregate function over the sequence of data, where the main point is to assume severe restrictions on the space usage. Note that in our setting, we focus on obtaining low time complexity per update and query, rather than optimizing the space complexity. In this respect, our work leans more towards the area of dynamic data structures, in particular dynamic query evaluation [9, 18].

For Boolean properties several papers [21, 22, 6] have considered streaming algorithms for testing membership in regular and context-free languages. Another variant of the problem was considered in [16, 15, 14], where the regular property is verified on the last N letters of the stream, instead of the entire prefix up to the current position.

The closest to our setting is the work [24], which studies the dynamic evaluation problem for monoids over a sliding window, and describes a data structure that can be updated in constant time for a fixed finite monoid. When the monoid is finite, the considered problem is basically the same as monitoring whether the input stream restricted to the sliding window is accepted by a finite automaton. We show in Example 1, that in this case, the problem can be reduced to the dynamic acceptance problem for a special form of timed automaton.

2 Preliminaries

Finite automata. A *finite automaton* is a tuple $\mathcal{A} = (\Sigma, Q, I, E, F)$, where Σ is a finite alphabet, Q is a finite set of states, $E \subseteq Q \times \Sigma \times Q$ is a transition relation, and $I, F \subseteq Q$ are the sets of initial and final states. A run of $\mathcal{A}$ on a word $w = a_1 \dots a_n \in \Sigma^*$ is a sequence $\rho = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} q_n$ where $(q_{i-1}, a_i, q_i) \in E$ for all $i = 1, \dots, n$. Moreover, ρ is a *successful* run if $q_0 \in I$ and $q_n \in F$. A word w is *accepted* by $\mathcal{A}$ if there is a successful run of $\mathcal{A}$ on w.

Timed automata. Let X be a finite set of clocks, usually denoted $\mathtt{x}, \mathtt{y}, \dots$. A *clock valuation* is a function $\nu : X \to \mathbb{R}_{\geq 0}$ from clocks to non-negative reals. *Clock conditions* are formulas defined by the grammar: $C_X \;:=\; \mathtt{true} \mid \mathtt{x} < c \mid \mathtt{x} > c \mid \mathtt{x} = c \mid C_X \wedge C_X \mid C_X \vee C_X$, where $\mathtt{x} \in X$ and $c \in \mathbb{R}_{\geq 0}$. By a slight abuse of notation, we also denote by C_X the set of clock conditions over X. Given a clock condition γ and a valuation ν, we say that ν *satisfies* γ and write $\nu \models \gamma$, if the arithmetic expression obtained from γ by substituting each clock $\mathtt{x}$ with its value $\nu(\mathtt{x})$ evaluates to true.

A *timed automaton* is a tuple $\mathcal{A} = (\Sigma, Q, X, I, E, F)$, where Q, Σ, I, F are defined exactly as for finite automata, X is a finite set of clocks, and $E \subseteq Q \times \Sigma \times C_X \times Q \times 2^X$ is a finite transition relation. We say that $c \in \mathbb{R}_{\geq 0}$ is a *clock constant* of $\mathcal{A}$ if c appears in some clock condition of a transition from E. A *configuration* of $\mathcal{A}$ is a pair (q, ν), where $q \in Q$ and ν is a clock valuation. Recall that finite automata process words over a finite alphabet Σ; likewise, timed automata process timed words over an alphabet of the form $\Sigma \uplus \mathbb{R}_{>0}$, with Σ finite.

A *run* of a timed automaton $\mathcal{A}$ on a timed word $w = e_1 \dots e_n \in (\Sigma \cup \mathbb{R}_{>0})^*$ is a sequence $\rho = (q_0, \nu_0) \xrightarrow{e_1} (q_1, \nu_1) \xrightarrow{e_2} \dots \xrightarrow{e_n} (q_n, \nu_n)$, where each (q_i, ν_i) is a configuration and

- if $e_i \in \mathbb{R}_{>0}$, then $q_{i+1} = q_i$ and $\nu_{i+1}(\mathtt{x}) = \nu_i(\mathtt{x}) + e_i$ for all $\mathtt{x} \in X$;
- if $e_i \in \Sigma$, then there is a transition $(q_i, e_i, \gamma, q_{i+1}, Z) \in E$ such that $\nu_i \models \gamma$ and either $\nu_{i+1}(\mathtt{x}) = 0$ or $\nu_{i+1}(\mathtt{x}) = \nu_i(\mathtt{x})$ depending on whether $\mathtt{x} \in Z$ or $\mathtt{x} \in X \setminus Z$.

Thus, the set Z in a transition $(q_i, e_i, \gamma, q_{i+1}, Z) \in E$ corresponds to the subset of clocks that are reset when firing the transition. Note that the values of the other clocks stay unchanged. An example of a one clock timed automaton was given in the introduction (see Figure 1).

A run ρ as above is *successful* if $q_0 \in I$, $\nu_0(\mathtt{x}) = 0$ for all $\mathtt{x} \in X$, and $q_n \in F$. A word $w \in (\Sigma \cup \mathbb{R}_{>0})^*$ is *accepted* by $\mathcal{A}$ if there is a successful run of $\mathcal{A}$ on w.

Size of an automaton. The size of a finite automaton $\mathcal{A} = (\Sigma, Q, I, E, F)$ is defined as $|\mathcal{A}| = |Q| + |E|$. This is asymptotically equivalent to essentially every possible definition of size of a finite automaton that can be found in the literature. The size of a timed automaton $\mathcal{A} = (\Sigma, Q, X, I, E, F)$ is instead defined as $|\mathcal{A}| = |Q| + |X| + \sum_{(p,a,\gamma,q,Z) \in E} |\gamma|$, where $|\gamma|$ is the number of atomic expressions (i.e. expressions of the form $\mathtt{true}$, $\mathtt{x} < c$, $\mathtt{x} > c$, $\mathtt{x} = c$)

appearing in the clock condition γ. *Note that the size of a timed automaton does not take into account the magnitude of the clock constants.* These constants are specified with the automaton and stored in suitable floating-point memory cells (see the computation model below).

Computation model. As clock constants and time spans in the input stream are arbitrary real numbers, it is convenient to use the *real RAM model* of computation. This is a standard model with integer memory cells that can store integers and floating-point memory cells that can store real numbers. There are no bounds on the bit length or precision of the stored numbers. Basic arithmetic operations – addition, subtraction, multiplication, and division – can be performed in unit time, but modulo arithmetics and rounding are not included in the model. In fact, we do not use multiplication or division on real numbers either.

3 The dynamic acceptance problem and main results

The *dynamic acceptance problem* amounts to designing a data structure that can be initialized for a given timed automaton $\mathcal{A}$ with one clock, and afterwards, upon consuming consecutive elements of the data stream, efficiently maintains the information on whether the word read so far is accepted by $\mathcal{A}$. Formally, the data structure should support the following operations:

- `init`($\mathcal{A}$): Initialize the data structure for a given automaton $\mathcal{A}$. This automaton is fixed for the entire lifespan of the data structure.
- `accepted`(): Query whether the prefix of the stream consumed up to the current moment is accepted by $\mathcal{A}$.
- `read`(e): Consume the next element e from the input stream, be it a letter from Σ or a time span from $\mathbb{R}_{>0}$, and update the data structure accordingly.

The running time of each of these operations needs to be as low as possible. More precisely, we shall say that a data structure *supports dynamic acceptance in time* $f(s,n)$ if the first operation `init`($\mathcal{A}$) takes at most $f(s,0)$ time, and every subsequent execution of `accepted`() or `read`(e) takes at most $f(s,n)$ time, where $s = |\mathcal{A}|$ and n is the number of stream elements consumed so far. Similarly, a data structure *supports dynamic acceptance in amortized time* $f(s,n)$ if the first operation `init`($\mathcal{A}$) takes at most $f(s,0)$ time and, for every n, the first n operations of the form `accepted`() and `read`(e) take at most $n \cdot f(s,n)$ time in total. Ultimately, we are interested in designing data structures where the complexity guarantee $f(s,n)$ is independent of n, that is, the (amortized) update time is a function of $|\mathcal{A}|$ only.

Before presenting the complexity results in detail, we provide an example of application of the dynamic acceptance problem.

▶ **Example 1.** We discuss the relationship between our dynamic acceptance problem for timed automata and an aggregation problem for monoids over a sliding window, as considered in [24]. When the monoid is finite, every element of it represents a regular language, and thus the aggregation problem can be seen as an acceptance problem. This means that the aggregation problem for finite monoids over a sliding window is reducible to an automaton membership problem in the *sliding window model* (see also [14]). We formalize this problem below.

Let $\mathcal{A} = (\Sigma, Q, I, E, F)$ be a finite automaton and C a positive integer defining the width of the sliding window. The membership problem of $\mathcal{A}$ with a sliding window of width C consists of processing, from left to right, an arbitrary input $w = a_1 a_2 a_3 \ldots$ over Σ, while maintaining the answer to the following query: *is the sequence of the last C consumed letters accepted by $\mathcal{A}$?* The goal is to design a data structure that can be updated in a time that only depends on the automaton $\mathcal{A}$, and not on the size of the window C.

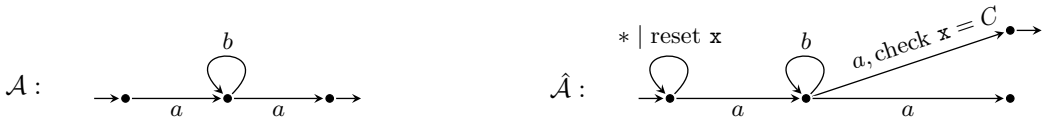


Figure 2 Reducing the sliding window membership problem to the dynamic acceptance problem.

Next, we explain how the above problem can be reduced to our dynamic acceptance problem. Here, we consider only streams that are discrete, and in fact even slightly more restricted: we assume that every input stream belongs to the language $(\{1\}\cdot\Sigma)^\omega$, namely, that the letters from Σ are interleaved by the time unit 1. We map the input word $w = a_1a_2a_3\ldots$ to a corresponding discrete stream $\widehat{w} = 1a_11a_21a_3\ldots$, and modify the finite automaton $\mathcal{A}$ to obtain a corresponding timed automaton $\widehat{\mathcal{A}}$, as follows. We introduce a new state $\widehat{q}$, which will be the only final state of $\widehat{\mathcal{A}}$, and a clock $\mathtt{x}$. We then replace every transition (q, a, q') of $\mathcal{A}$ with the transition $(q, a, \mathtt{true}, q', \emptyset)$. Note that these transitions have a vacuous clock condition, hence they are applicable in $\widehat{\mathcal{A}}$ whenever the original transitions of $\mathcal{A}$ are so. In addition, when the former transition (q, a, q') reaches a final state $q' \in F$, we also have a transition $(q, a, \mathtt{x} = C, \widehat{q}, \emptyset)$ in $\widehat{\mathcal{A}}$. Finally, we add looping transitions on the initial states that reset the clock, that is, transitions of the form $(q, a, \mathtt{true}, q, \{\mathtt{x}\})$, with $q \in I$ and $a \in \Sigma$. Figure 2 shows the timed automaton $\widehat{\mathcal{A}}$ corresponding to an automaton $\mathcal{A}$ recognising ab^*a.

From the above construction it is clear that $\widehat{\mathcal{A}}$ accepts a prefix $1a_1\ldots1a_n$ of $\widehat{w}$ if and only if $\mathcal{A}$ accepts the C-letter factor $a_{n-C+1}\ldots a_n$ of w. Thus, the membership problem for $\mathcal{A}$ in the C-width sliding window model is reduced to the dynamic acceptance problem for $\widehat{\mathcal{A}}$ over the stream $\widehat{w}$. We will see later (Theorem 2) that there is a data structure that supports dynamic acceptance for $\widehat{\mathcal{A}}$ with update time $2^{\mathcal{O}(|\widehat{\mathcal{A}}|)} = 2^{\mathcal{O}(|\mathcal{A}|)}$. This means that we can process one letter at a time from a word w, while answering in time $2^{\mathcal{O}(|\mathcal{A}|)}$ whether $\mathcal{A}$ accepts the sequence of the last C consumed letters. Note that the complexity here is independent of C.

Results. We say that a stream w is *discrete* if its elements range over $\Sigma \uplus \{1\}$, that is, if all time spans in the stream coincide with the time unit 1. Our main result is the following:

▶ **Theorem 2.** *Consider the dynamic acceptance problem for timed automata with one clock. There is a data structure that*

- *supports dynamic acceptance in time $2^{\mathcal{O}(|\mathcal{A}|)}$ on discrete streams, and*
- *supports dynamic acceptance in amortized time $2^{\mathcal{O}(|\mathcal{A}|)}$ on arbitrary streams,*

where $\mathcal{A}$ is the automaton provided upon initialization.

We stress that the complexity in Theorem 2 depends only on the size of $\mathcal{A}$. In particular, it does not depend on the bitlength of clock constants (e.g. 24 in Figure 1). Note that thanks to the assumption of the real RAM model, the question of the complexity of arithmetic operations on reals is separated from the running time analysis in the proof of Theorem 2. This feature reflects the real-life scenarios, where the automaton is small, while real numbers involved can be efficiently manipulated by the processor despite having large bitlength. The proof of Theorem 2 is presented in Section 4.

We do not know whether this theorem can be generalized to timed automata with more than one clock while preserving independence of the time complexity of updates from the length of the consumed stream prefix. However, we establish a negative result for a slightly more powerful model of timed automata, called timed automata with additive constraints (see e.g. [8]). Formally, a *timed automaton with additive constraints* is defined exactly as a

timed automaton – that is, as a tuple $\mathcal{A} = (\Sigma, Q, X, I, E, F)$ consisting of an input alphabet, a set of states, a set of clocks, etc. – but clock conditions are now allowed to satisfy an extended grammar obtained by adding new rules of the form $\left(\sum_{\mathtt{x}\in Z} \mathtt{x}\right) \sim c$, where $Z \subseteq X$ and $\sim \in \{<, >, =\}$. For instance, one can write $\mathtt{x} + \mathtt{y} \leq c$, where c is a clock constant. To give some background, let us briefly discuss in more detail the power of this extension. Allowing additive constraints is a non-trivial extension of timed automata and in particular it makes the emptiness problem undecidable [8, Theorem 2]. However, undecidability holds when at least four clocks are available. Moreover, it is shown that for timed automata with additive constraints with two clocks the emptiness problem is decidable; and the proof is a straightforward modification of the standard region construction [8, Proposition 1].

Our negative result relies on the 3SUM conjecture, stated just below. Recall that in the 3SUM problem we are given a set S of positive real numbers and the question is to determine whether there exist $a, b, c \in S$ satisfying $a + b = c$. It is easy to solve the problem in time $\mathcal{O}(n^2)$, where $n = |S|$; the 3SUM Conjecture asserts that this cannot be significantly improved:

▶ **Conjecture 3** (3SUM Conjecture)**.** *In the real RAM model, the* 3SUM *problem cannot be solved in strongly sub-quadratic time, that is, in time $\mathcal{O}(n^{2-\delta})$ for any $\delta > 0$, where n is the number of values forming the input.*

The 3SUM Conjecture is widely used in computational geometry and fine-grained complexity theory (see an overview in [2, Appendix A]), and it was applied to establish lower bounds for several dynamic problems [1, 3, 19, 23]. Our negative result is similar in nature:

▶ **Theorem 4.** *If the* 3SUM *Conjecture holds, then there is a two-clock timed automaton $\mathcal{A}$ with additive constraints such that there is no data structure that, when initialized on $\mathcal{A}$, supports dynamic acceptance in time $\mathcal{O}(n^{1-\delta})$ for any $\delta > 0$, where n is the length of the consumed stream prefix.*

The proof of Theorem 4 follows almost directly from an analogous 3SUM-hardness result in the static setting:

▶ **Lemma 5.** *If the* 3SUM *Conjecture holds, then there is a two-clock timed automaton $\mathcal{A}$ with additive constraints for which there is no algorithm that, given a finite timed word $w \in \left(\Sigma \uplus \mathbb{R}_{>0}\right)^*$ as input, where Σ is a two-letter alphabet, decides whether $\mathcal{A}$ accepts w in time $\mathcal{O}(n^{2-\delta})$ for any $\delta > 0$ and for $n = |w|$.*

Proof. We construct a two-clock timed automaton $\mathcal{A}$ with additive constraints and an algorithm that given a set S of n positive reals, outputs a word $w \in \left(\Sigma \uplus \mathbb{R}_{>0}\right)^*$ such that w is accepted by $\mathcal{A}$ if and only if there are $a, b, c \in S$ satisfying $a + b = c$. We find it more convenient to first present the construction of w from S. Then we present the automaton $\mathcal{A}$ and analyze its runs on w.

Let $S = \{s_1, s_2, \ldots, s_n\}$ be a set of positive real numbers and $M = \max(S) + 1$. By sorting S we may assume that $0 < s_1 < \ldots < s_n < M$. We set $\Sigma = \{\diamondsuit, \spadesuit\}$. The word is defined as

$$w = u \spadesuit u \spadesuit v,$$

where

$$\begin{aligned} u &= 2(M - s_n) \diamondsuit 2(s_n - s_{n-1}) \diamondsuit 2(s_{n-1} - s_{n-2}) \diamondsuit \ \ldots \ \diamondsuit 2(s_2 - s_1) \diamondsuit 2(s_1 - 0); \\ v &= (M - s_n) \diamondsuit \ (s_n - s_{n-1}) \diamondsuit \ (s_{n-1} - s_{n-2}) \diamondsuit \ \ldots \ \diamondsuit \ (s_2 - s_1) \diamondsuit. \end{aligned}$$

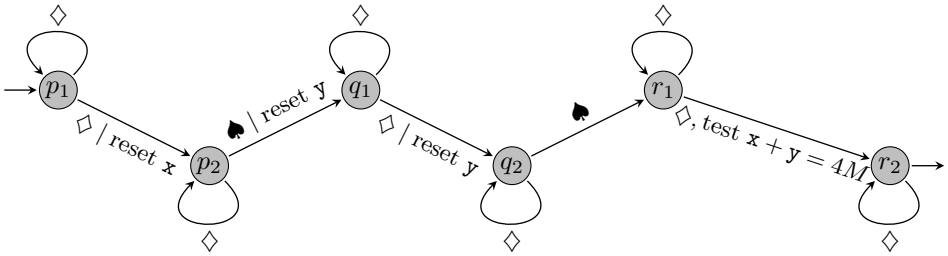


Figure 3 Timed automaton for reducing 3SUM.

Note that w has length $\mathcal{O}(n)$ and can be constructed from S in time $\mathcal{O}(n \log n)$. Intuitively, the factors u, u, and v above are responsible for the choice of a, b, and c, respectively.

We now describe a timed automaton $\mathcal{A}$ that accepts w if and only if $a + b = c$. The automaton is depicted in Figure 3. It uses two clocks, named $\mathtt{x}$ and $\mathtt{y}$. All the transitions have trivial (always true) clock conditions, apart from the transition from r_1 to r_2, where we check that the sum of clock values is equal to $4M$. The only initial state is p_1; the only accepting state is r_2.

Next, we analyze the runs of $\mathcal{A}$ on w, with the goal of showing that $\mathcal{A}$ accepts w if and only if there are $a, b, c \in S$ such that $a + b = c$. Consider any successful run ρ of $\mathcal{A}$ on w. Observe that the moment of reading the first symbol ♠ in w must coincide with firing the transition from p_2 to q_1. At this moment, the automaton has consumed the first factor u of w, and there was a moment where it moved from state p_1 to state p_2 upon reading one of the ♢ symbols from u. Supposing that the transition in ρ from p_1 to p_2 happens at the i-th symbol ♢ of u, the clock valuation at the moment of reaching q_1 for the first time must satisfy $\mathtt{x} = 2(s_i - s_{i-1}) + \ldots + 2(s_2 - s_1) + 2s_1$ $(= 2s_i)$ and $\mathtt{y} = 0$. We conclude the following.

▷ Claim 6. The set of possible clock valuations at the moment of reaching the state q_1 for the first time is $\{(\mathtt{x} = 2a,\ \mathtt{y} = 0) \ :\ a \in S\}$.

Next, observe that the moment of reading the second occurrence of ♠ in w must coincide with firing the transition from q_2 to r_1. Between the first and the second symbol ♠ the automaton consumes the second factor u, and during this the clock $\mathtt{x}$ increases exactly by the sum of the time spans within u, i.e. by $2M$. On consuming the second factor u, the clock $\mathtt{y}$ is reset once, and precisely when firing the transition from q_1 to q_2, which happens on reading one of the occurrences of ♢ in u. Again, if this happens when reading the j-th occurrence of ♢, then, after the reset, $\mathtt{y}$ is incremented by exactly $2s_j$ units. We conclude the following.

▷ Claim 7. The set of possible clock valuations at the moment of reaching the state r_1 for the first time is $\{(\mathtt{x} = 2a + 2M,\ \mathtt{y} = 2b) \ :\ a, b \in S\}$.

Finally, after consuming the last factor v, the automaton can move to the accepting state r_2 if and only if at some point, upon reading an occurrence of ♢, the condition $x + y = 4M$ holds. Observe that the sum of the first k numbers encoded in v is equal to $M - s_{n-k+1}$. Hence, after parsing those numbers, the set of possible clock valuations is $\{(\mathtt{x} = 2a + 2M + M - c,\ \mathtt{y} = 2b + M - c) \ :\ a, b \in S\}$, for some choice of $c \in S$. Moreover, the latter valuations satisfy the condition $\mathtt{x} + \mathtt{y} = 4M$ if and only if $a + b = c$.

Based on the above arguments, we infer that a successful run like ρ exists on input w if and only if there are $a, b, c \in S$ such that $a + b = c$. To conclude the proof, we observe that if an algorithm could decide whether $\mathcal{A}$ accepts w in time $\mathcal{O}(n^{2-\delta})$ for any $\delta > 0$, then by combining this algorithm with the presented construction, one could solve 3SUM in time $\mathcal{O}(n^{2-\delta})$. This would contradict the 3SUM Conjecture. ◀

We conclude the section by showing how Theorem 4 follows from Lemma 5. Consider the timed automaton $\mathcal{A}$ provided by the lemma. If a data structure as in the statement of the theorem existed, then using this data structure one could decide in strongly sub-quadratic time whether any input timed word w is accepted by $\mathcal{A}$, by simply applying the sequence of `read`$(\cdot)$ operations corresponding to w, followed by the query `accepted`$()$.

Recall that we do not know whether a negative result similar to Theorem 4 also holds for plain timed automata (without additive constraints).

4 Data structure: proof of Theorem 2

Notation. Let us fix, once and for all, the timed automaton $\mathcal{A} = (\Sigma, Q, X, I, E, F)$ with a single clock `x` that is provided upon initialization. By adding a non-accepting sink state, if necessary, we may assume that for every $q \in Q$ and $a \in \Sigma$, some transition over letter a can be always applied at q at any time. As $\mathcal{A}$ uses only one clock, every configuration of $\mathcal{A}$ can be written simply as a pair (q, t), where $q \in Q$ is the state and $t \in \mathbb{R}_{\geq 0}$ is the value of the clock `x`.

Let $0 = C_0 < C_1 < \ldots < C_m$ be the clock constants used in $\mathcal{A}$, where we assume without loss of generality that $C_0 = 0$. For simplicity we also let $C_{m+1} = \infty$. Note that $m \leq |\mathcal{A}|$.

Consider now an arbitrary stream $w \in (\Sigma \cup \mathbb{R}_{>0})^\omega$. For every $n \in \mathbb{N}$, let $w_n = w[1 \ldots n]$ be the n-element prefix of w. Recall that w_n can be thought of as the stream prefix that is disclosed after n operations `read`(e). We say that a configuration (q, t) is *active* at step n if there is a run of $\mathcal{A}$ on w_n that starts in a configuration $(q_0, 0)$ for some $q_0 \in I$ and ends in (q, t). We let K_n be the set of all configurations (q, t) that are active at step n.

Partitioning the problem. It is clear that the dynamic acceptance problem essentially boils down to designing an efficient data structure that maintains K_n upon reading subsequent elements from the stream. This data structure should offer a query on whether K_n contains an accepting configuration. The main observation is that configurations with clock values that are in the same order with respect to the clock constants $C_1, \ldots, C_m$ satisfy exactly the same clock conditions in E. Precisely, let us consider the partition of $\mathbb{R}_{\geq 0}$ into intervals $J_0,\ J_1,\ \ldots J_{2m+1}$, where $J_{2i} = [C_i, C_i]$, $J_{2i+1} = (C_i, C_{i+1})$, for all $i \in \{0, \ldots, m\}$. The following assertion holds: for any two configurations (q, t), (q, t'), with $t, t' \in J_i$ for some $0 \leq i \leq 2m + 1$, exactly the same transitions are available in (q, t) as in (q, t').

For $n \in \mathbb{N}$ and $i \in \{0, \ldots, 2m + 1\}$, let

$$K_n[i] = \{\, (q, t) \in K_n \ :\ t \in J_i \,\}.$$

The idea is to maintain each set $K_n[i]$ in a separate data structure. Each of these data structures follows the same design, which we call the *inner data structure*.

Inner data structure: an overview. Every inner data structure is constructed for an interval $J \in \{J_0, \ldots, J_{2m+1}\}$. We will denote it by $\mathbb{D}[J]$, or simply by $\mathbb{D}[i]$ when $J = J_i$. Each structure $\mathbb{D}[J]$ stores a set of configurations L satisfying the following invariant: all clock values of configurations in L belong to J. In the final design we will maintain the invariant that the set L stored by $\mathbb{D}[i]$ at step n is equal to $K_n[i]$, but for the design of $\mathbb{D}[J]$ it is easier to treat L as an arbitrary set of configurations with clock values in J.

The inner data structure should support the following methods:

- Method `init`(J) stores the interval J and initializes $\mathbb{D}[J]$ by setting $L = \emptyset$.

- Method `accepted`() returns true or false, depending on whether or not L contains an accepting configuration, that is, a configuration (q, t) such that $q \in F$.
- Method `insert`(q, t) adds a configuration (q, t) to L. This method will be always applied with a promise that $t \in J$ and $t \leq t'$ for all configurations (q', t') already present in L.
- Method `updateTime`(r), where $r \in \mathbb{R}_{>0}$, increments the clock values of all configurations in L by r. All configurations whose clock values ceased to belong to J are removed from L, and they are returned by the method on output. This output is organised as a doubly linked list of configurations, sorted by non-decreasing clock values.
- Method `updateLetter`(a) updates L by applying to all configurations in L all possible transitions over the given letter $a \in \Sigma$. Precisely, the updated set comprises all configurations (q, t) that can be obtained from configurations belonging to L before the update using transitions over a that do not reset the clock. The configurations $(q, 0)$ which can be obtained from L using transitions over a that do reset the clock are not included in the updated set, but are instead returned by the method as a doubly linked list.

In Section 4.2 we will provide an efficient implementation of the inner data structure, which is encapsulated in the following lemma.

▶ **Lemma 8.** *For each $J \in \{J_0, J_1, \ldots, J_{2m+1}\}$, the inner data structure $\mathbb{D}[J]$ can be implemented so that methods* `init`(), `accepted`(), `insert`$(\cdot,\cdot)$, *and* `updateLetter`$(\cdot)$ *run in time $2^{\mathcal{O}(|\mathcal{A}|)}$, while method* `updateTime`$(\cdot)$ *runs in time $2^{\mathcal{O}(|\mathcal{A}|)} \cdot \ell$, where ℓ is the size of its output.*

We postpone the proof of Lemma 8 and we show now how to use it to prove Theorem 2. That is, we design an *outer data structure* that monitors the acceptance of $\mathcal{A}$.

4.1 Outer data structure

The outer data structure consists of a list $\mathbb{D}[0], \ldots, \mathbb{D}[2m+1]$, where each $\mathbb{D}[i]$ is a copy of the inner data structure constructed for the interval J_i. We will keep the following invariant:

I1. After step n, for each $i \in \{0, 1, \ldots, 2m+1\}$ the data structure $\mathbb{D}[i]$ stores $K_n[i]$.

We first explain how the outer data structure implements the promised operations: initialization, queries about the acceptance, and updates upon reading the next element of the stream w. Then we discuss the amortized complexity of the updates.

Initialization. Given $\mathcal{A}$, we store $\mathcal{A}$ in the data structure and we read the clock constants $0 = C_0 < C_1 < \ldots < C_m$ from $\mathcal{A}$. Then we initialize $2m+1$ copies $\mathbb{D}[0], \ldots, \mathbb{D}[2m+1]$ of the inner data structure by calling method `init`(J) for each interval J among $J_0, J_1, \ldots, J_{2m+1}$. Finally, for each initial state q, we apply method `insert`$(q, 0)$ on $\mathbb{D}[0]$. As $K_0 = \{(q, 0) : q \in I\}$, after this we have that Invariant (I1) holds for $n = 0$.

Query. We query all the data structures $\mathbb{D}[0], \ldots, \mathbb{D}[2m+1]$ for the existence of accepting configurations using the `accepted`() method, and return the disjunction of the answers. The correctness follows directly from Invariant (I1).

Update by a time span. Suppose the next element from the stream is a time span $r \in \mathbb{R}_{>0}$. We update the outer data structure as follows. First, we apply method `updateTime`(r) to each data structure $\mathbb{D}[i]$. This operation increments the clock values of all configurations stored in $\mathbb{D}[i]$ by r, but may output a set of configurations whose clock values ceased to fit in

the interval J_i. Recall that this set is organised as a doubly linked list of configurations, sorted by non-decreasing clock values; call this list S_i. Now, we need to insert each configuration (q,t) that appears on those lists into the appropriate data structure $\mathbb{D}[j]$, where j is such that $t \in J_j$. However, we have to be careful about the order of insertions: we process the lists $S_{2m+1}, S_{2m}, \ldots, S_0$ in this precise order, and each list S_i is processed from the end, that is, following the non-increasing order of clock values. When processing a configuration (q,t) from the list S_i, we find the index $j > i$ such that $t \in J_j$ and apply the method $\texttt{insert}(q,t)$ on the structure $\mathbb{D}[j]$. In this way the condition required by the `insert` method – that $t \leq t'$ for every configuration (q',t') currently stored in $\mathbb{D}[j]$ – is satisfied. It is also easy to see that Invariant (I1) is preserved after the update.

Update by a letter. Suppose the next symbol read from the stream is a letter $a \in \Sigma$. We update the outer data structure as follows. First, we apply method $\texttt{updateLetter}(a)$ to each data structure $\mathbb{D}[i]$. This operation applies all possible transitions on letter a to all configurations stored in $\mathbb{D}[i]$, and outputs a list of configurations R_i where the clock got reset. All these configurations have clock value 0, hence the length of R_i is at most $|Q|$. It now suffices to insert all the configurations $(q,0)$ appearing on all the lists R_i to $\mathbb{D}[0]$ using method $\texttt{insert}(q,0)$. We may do this in any order, as the condition required by the `insert` method is trivially satisfied. Again, Invariant (I1) is clearly preserved after the update.

This concludes the implementation of the outer data structure. While the correctness is clear from the description, we are left with arguing that the time complexity is as promised.

From Lemma 8 it readily follows that each of the following operations takes time $2^{\mathcal{O}(|\mathcal{A}|)}$: initialization, a query about the acceptance, and an update by a letter. As for an update by a time span $r \in \mathbb{R}_{>0}$, by Lemma 8 the complexity of such an update is $2^{\mathcal{O}(|\mathcal{A}|)} \cdot \sum_{i=0}^{2m+1} |S_i|$, where $S_0, \ldots, S_{2m+1}$ are the sets returned by the applications of method $\texttt{updateTime}(r)$ to data structures $\mathbb{D}[0], \ldots, \mathbb{D}[2m+1]$, respectively. We need to argue that the amortized time complexity of all these updates is bounded by $2^{\mathcal{O}(|\mathcal{A}|)}$.

Consider the following definition: a clock value $t \in \mathbb{R}_{\geq 0}$ is *active* at step n if K_n contains a configuration with clock value t. Observe that upon an update by a time span $r \in \mathbb{R}_{>0}$, the set of active clock values simply gets shifted by r, while upon an update by a letter $a \in \Sigma$ it stays the same, except that clock value 0 may also become active. Since at step 0 the only active clock value is 0, we conclude that for every $n \in \mathbb{N}$, at most $n+1$ active clock values may have appeared until step n. Note that there may be at most $|Q|$ different active configurations with the same active clock value, hence the complexity of each update by a time span is bounded by $2^{\mathcal{O}(|\mathcal{A}|)} \cdot |Q|$ times the number of active clock values that change membership from an interval to another one, where we imagine that each active clock value is shifted by the time span. As every active clock value can change membership in an interval at most $2m+1$ times, and since the total number of active values that appear until step n is at most $n+1$, we conclude that the total time spent on updates by time spans throughout the first n steps is bounded by $2^{\mathcal{O}(|\mathcal{A}|)} \cdot |Q| \cdot (2m+1) \cdot (n+1) = 2^{\mathcal{O}(|\mathcal{A}|)} \cdot n$. Hence, the amortized time complexity is $2^{\mathcal{O}(|\mathcal{A}|)}$.

Finally, note that in the case of discrete streams each set S_i consists of configurations with the same clock value, hence $|S_i| \leq |Q| \leq |\mathcal{A}|$ for all $i \in \{0, \ldots, 2m+1\}$. So in this case, the complexity of an update by a time span is bounded by $2^{\mathcal{O}(|\mathcal{A}|)}$, without any amortization.

This finishes the proof of Theorem 2, assuming Lemma 8. We prove the latter next.

4.2 Inner data structure

We now describe the inner data structure $\mathbb{D}[J]$ and prove Lemma 8. Let us fix an interval $J \in \{J_0, \ldots, J_{2m+1}\}$. We denote by L the set of configurations currently stored by the inner data structure $\mathbb{D}[J]$. It is convenient to represent L by a function $\lambda\colon \mathbb{R}_{\geq 0} \to 2^Q$ defined by

$$\lambda(t) = \{\, q \in Q \,:\, (q,t) \in L \}.$$

We let $\widehat{L}$ be the set of all clock values that are *active* in L, that is, $\widehat{L}$ comprises all $t \in \mathbb{R}_{\geq 0}$ such that $\lambda(t) \neq \emptyset$. Recall that we assume that $\widehat{L} \subseteq J$.

Before we dive into the details, let us discuss the intuition. The basic idea is to store all the configurations in L in a queue, implemented as a doubly-linked list ordered by non-decreasing clock values. To handle clock values efficiently, we do not store them directly. Instead, we maintain a global clock that measures the total time since the initialization of the data structure, and each configuration bears a timestamp that is the value of this global clock at the moment of the last reset. Thus, updating by a time span boils down to increasing the value of the global clock and popping any configurations at the back of the queue whose clock values ceased to fit into the interval J.

Updating by a letter is more problematic, as we need to apply the transition relation of the automaton $\mathcal{A}$ to all the configurations of L simultaneously. In the data structure we store a partition of the active clock values $\widehat{L}$ according to their images under $\lambda(\cdot)$, so that for each block of this partition (whose number is at most $2^{|Q|}$), we can simultaneously update all corresponding configurations in constant time. There is a caveat here: it is possible that for some $t, t' \in \widehat{L}$ we have $\lambda(t) \neq \lambda(t')$ before the update, but $\lambda(t) = \lambda(t')$ after the update. That is, the blocks of the partition may require merging upon updates. We resolve this issue by representing the partition in a *forest*, similarly as the union-find data structure would do. The key point is that the height of this forest can be kept bounded by $2^{|Q|}$.

Description of the structure. In short, the data structure $\mathbb{D}[J]$ consists of three elements:

- The *clock*, denoted `y`, is a real that represents the total time elapsed since initialization.
- The *list*, denoted `l`, stores the set of active clock values $\widehat{L}$.
- The *forest*, denoted `f`, is built on top of the elements of `l` and describes the function λ.

We describe the list and the forest in more details (the reader can refer to Figure 4).

The list. The list `l` encodes the clock values present in $\widehat{L}$, sorted in the increasing order and organised into a doubly linked list. Each node α on `l` is a record consisting of:

- $\texttt{next}(\alpha)$: a pointer to the next node on the list;
- $\texttt{prev}(\alpha)$: a pointer to the previous node on the list; and
- $\texttt{timestamp}(\alpha) \in \mathbb{R}$: the *timestamp* of the node.

As usual, the data structure stores `l` by maintaining pointers to the first and last nodes.

The clock value represented by a node α on `l` is equal to $\texttt{clock}(\alpha) = \texttt{y} - \texttt{timestamp}(\alpha)$; this will always be a non-negative real. Thus, the timestamp is essentially the total elapsed time recorded at the moment of the last reset of the clock. Note that this implementation allows for a simultaneous increment of $\texttt{clock}(\alpha)$ for all nodes α on `l` in constant time: it suffices to simply increment `y`.

The forest. Forest `f` represents the mapping from elements $t \in \widehat{L}$, encoded in `l`, to respective sets of control states $\lambda(t)$. It is a rooted forest where nodes may have arbitrarily many children, and these children are unordered. Every node γ of `f` is a record containing:

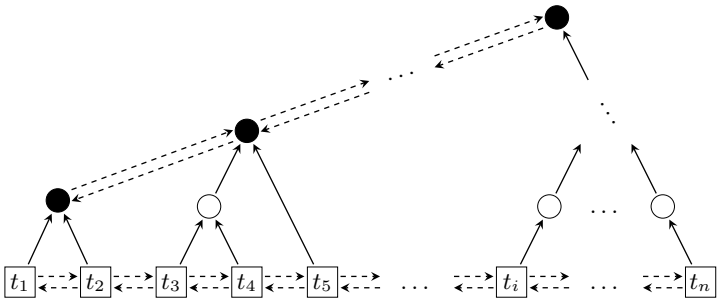


Figure 4 The inner data structure. List elements are depicted as squares while the forest nodes are depicted as circles. The black circles are the roots.

- $\texttt{parent}(\gamma)$: a pointer to the parent of γ; and
- $\#\texttt{children}(\gamma)$: an integer equal to the number of children of γ.

The leaves of the forest will always coincide with the nodes on the list `l`. In particular, we augment the records stored for the nodes on `l` by adding the $\texttt{parent}(\cdot)$ pointer, and treat them as nodes of the forest `f` at the same time. The counter $\#\texttt{children}(\cdot)$ would always be equal to 0 for those nodes, so we may omit it.

The *roots* of the forest are the nodes β with no parent, i.e. $\texttt{parent}(\beta) = \bot$. We will maintain the invariant that no root is a leaf in `f`, that is, every root has at least one child. In the data structure we store a doubly linked list containing all the roots of `f`. This list will be denoted `r`, and again it is stored by pointers to its first and last element. Thus, the records of the roots of `f` are augmented by $\texttt{next}(\cdot)$ and $\texttt{prev}(\cdot)$ pointers describing the structure of `r`, with the usual meaning. In addition to this, every root β of `f` carries two additional values:

- $\texttt{states}(\beta) \subseteq Q$: a non-empty subset of control states for which β is responsible; and
- $\texttt{rank}(\beta)$: an integer from the set $\{1, 2, 3, \ldots, 2^{|Q|}\}$.

We will maintain two invariants about these values. First, the sets $\texttt{states}(\beta)$ must be different for distinct roots β of `f`, and the same holds for the ranks $\texttt{rank}(\beta)$. Note that this implies that `f` has at most $2^{|Q|} - 1$ roots. Second, for every root β, the *tree rooted at* β – which is the tree containing β and all its descendants in `f` – has depth at most $\texttt{rank}(\beta) + 1$ – where the *depth* of a forest is the maximum number of edges on a path from a leaf to a root. Note that this implies that the depth of the forest `f` is bounded by $2^{|Q|} + 1$.

Function λ is then represented as follows. For every node α on `l`, let $\texttt{root}(\alpha)$ be the root of the tree of `f` that contains α. Then denoting $t = \texttt{clock}(\alpha)$, we have $\lambda(t) = \texttt{states}(\texttt{root}(\alpha))$. Note that the invariant stated above implies that from every leaf α of `f`, $\texttt{root}(\alpha)$ can be computed from α by following the $\texttt{parent}(\cdot)$ pointer at most $2^{|Q|}$ times. Hence, given $t \in \widehat{L}$ and a node α on `l` satisfying $t = \texttt{clock}(\alpha)$, we can compute $\lambda(t)$ in time $\mathcal{O}(2^{|Q|}) \leq 2^{\mathcal{O}(|\mathcal{A}|)}$.

Invariants. For convenience, we gather here all the invariants maintained by the inner data structure which we mentioned before:

I2. For each node α on `l`, the value $\texttt{clock}(\alpha) = \texttt{y} - \texttt{timestamp}(\alpha)$ belongs to J.

I3. The nodes on `l` are sorted by increasing clock values, or equally by decreasing timestamps. That is, $\texttt{timestamp}(\alpha) > \texttt{timestamp}(\texttt{next}(\alpha))$ for every non-last node α on `l`.

I4. Every root of `f` has at least one child, and the leaves of `f` are exactly all the nodes on `l`.

I5. The roots of `f` carry pairwise different, non-empty sets of control states, and they have pairwise different ranks. Moreover, all the ranks belong to the set $\{1, 2, \ldots, 2^{|Q|}\}$.

I6. For every root β of `f`, the depth of the tree rooted at β is at most $\texttt{rank}(\beta) + 1$.

Implementation. Now we show how to implement the methods `init`(J), `accepted`(), `insert`(q,t), `updateTime`(r), and `updateLetter`(a) in the data structure. Recall that all these methods should work in time $2^{\mathcal{O}(|\mathcal{A}|)}$, with the exception of `updateTime`(r) which is allowed to work in time $2^{\mathcal{O}(|\mathcal{A}|)} \cdot \ell$, where ℓ is the size of its output. The description of each method is supplied by a running time analysis and an argumentation of the correctness, which includes a discussion on why the invariants stated above are maintained.

Removing nodes. Before we proceed to the description of the required methods, we briefly discuss an auxiliary procedure of removing a node from the list `l` and from the forest `f`, as this procedure will be used several times. Suppose we are given a node α on the list `l` and we would like to remove it, which corresponds to removing from L all configurations (q,t) where $t = \texttt{clock}(\alpha)$ and $q \in \lambda(t)$. We can remove α from `l` in the usual way. Then we remove α from `f` as follows. First, we decrement the counter of children in the parent of α. If this counter stays positive then there is nothing more to do. Otherwise, we need to remove the parent of α as well, and accordingly decrement the counter of children in the grandparent of α. This can again trigger removal of the grandparent and so on. If eventually we need to remove a root of `f`, we also remove it from the list `r` in the usual way. Note that since by Invariants (I5) and (I6), the depth of `f` is bounded by $2^{|Q|}+1$, the whole procedure can be performed in time $\mathcal{O}(2^{|Q|}) \leq 2^{\mathcal{O}(|\mathcal{A}|)}$. It is clear that all the invariants are maintained.

Initialization. The `init`(J) method stores the interval J, that defines the range of clock values that could be represented in the data structure. It also sets `y` $= 0$ and initializes `l` and `r` as empty lists. The correctness and the running time are clear.

Acceptance query. The `accepted`() method is implemented as follows. We iterate through the list `r` to check whether there exists a root β of `f` such that `states`(`f`) contains any accepting state, say q. If this is the case, then by Invariant (I4) there is a node α on `l` satisfying `root`$(\alpha) = \beta$, hence (q,t) is an accepting configuration that belongs to L, where $t = \texttt{clock}(\alpha)$. So we may return a positive answer from the query. Otherwise, all configurations in L have non-accepting states, and we may return a negative answer. Note that since by Invariant (I5) the list `r` has length at most $2^{|Q|}-1$, the above procedure works in time $2^{\mathcal{O}(|\mathcal{A}|)}$.

Insertion. We now implement the method `insert`(q,t), where (q,t) is a configuration. Recall that when this method is executed, we have a promise that $t \in J$ and $t \leq t'$ for all configurations (q',t') that are currently present in $\mathbb{D}[J]$.

Let α be the first node on the list `l`. Let $t' = \texttt{clock}(\alpha)$. By the promise, we have $t \leq t'$. We consider cases: either $t < t'$ or $t = t'$. The former case also captures the situation when `l` is empty. When $t < t'$ or `l` is empty, the new configuration (q,t) gives rise to a new active clock value t. Therefore, we create a new list node α_0 and insert it at the front of the list `l`. We set the timestamp as `timestamp`$(\alpha_0) =$ `y` $- t$, so that the node correctly represents the clock value t. It is clear that Invariants (I2) and (I3) are thus satisfied.

Next, we need to insert the new node α_0 to the forest `f`. We iterate through the list `r` in search for a root β that satisfies `states`$(\beta) = \{q\}$. In case there is one, we simply set `parent`$(\alpha_0) = \beta$ and increment #`children`(β). Otherwise, we construct a new root β_0 with `states`$(\beta_0) = \{q\}$ and #`children`$(\beta_0) = 1$, insert it at the front of the list `r`, and set `parent`$(\alpha_0) = \beta_0$. To determine the rank of β_0, we find the smallest integer $k \in \{1, \ldots, 2^{|Q|}\}$ that is *not* used as the rank of any other root of `f`. Observe that, by Invariant (I5), the forest

$\mathtt{f}$ has at most $2^{|Q|} - 1$ roots, so there is always such a number k, and it can be found in time $2^{\mathcal{O}(|\mathcal{A}|)}$ by inspecting the list $\mathtt{r}$. We then set $\mathtt{rank}(\beta_0) = k$. It is clear that this operation can be performed in time $2^{\mathcal{O}(|\mathcal{A}|)}$, and that Invariants (I4), (I5), and (I6) are maintained. For the last one, observe that the new leaf α_0 is attached directly under a root of $\mathtt{f}$, so no tree in $\mathtt{f}$ existing before the insertion could have increased its depth.

We are left with the case when $t = t'$. We first compute the set X equal to $\lambda(t)$ before the insertion: it suffices to find $\mathtt{root}(\alpha)$ in time $2^{\mathcal{O}(|\mathcal{A}|)}$ and read $X = \mathtt{states}(\mathtt{root}(\alpha))$. If $q \in X$ then the configuration (q, t) is already present in L, so there is nothing to do. Otherwise, we need to update the data structure so that $\lambda(t)$ is equal to $X \cup \{q\}$ instead of X. Consequently, we remove the node α from $\mathtt{l}$ and from $\mathtt{f}$, using the operation described earlier, and we insert a new node α' at the front of $\mathtt{l}$, with the same timestamp equal to that of α. Thus, $\mathtt{clock}(\alpha') = t$. We next insert the new node α' to the forest $\mathtt{f}$ using the same procedure as described in the previous paragraph, but applied to the state set $X \cup \{q\}$ instead of $\{q\}$. Again, it is clear that these operations can be performed in time $2^{\mathcal{O}(|\mathcal{A}|)}$, and the same argumentation shows that all the invariants are maintained.

Update by a time span. Next, we implement the method $\mathtt{updateTime}(r)$, for $r \in \mathbb{R}_{>0}$. First, we increment $\mathtt{y}$ by r. Thus, for every node α in the list $\mathtt{l}$, the value $\mathtt{clock}(\alpha)$ is incremented by r. However, the Invariant (I2) may have ceased to hold, as some active clock values could have been shifted outside of the interval J. The configurations with these clock values should be removed from the data structure and their list should be the output of the method.

We extract these configurations as follows. Construct an initially empty list of configuration $\mathtt{lret}$, on which we shall build the output. Iterate through the list $\mathtt{l}$, starting from its back. For each consecutive node α, compute $t = \mathtt{clock}(\alpha)$. If $t \in J$, then break the iteration and return $\mathtt{lret}$, as there are no more configurations to remove. Otherwise, find $\mathtt{root}(\alpha)$ in time $2^{\mathcal{O}(|\mathcal{A}|)}$, read $\lambda(t) = \mathtt{states}(\mathtt{root}(\alpha))$, and add at the front of $\mathtt{lret}$ all configurations (q, t) for $q \in \lambda(t)$, in any order. Then remove α from the list $\mathtt{l}$ and from the forest $\mathtt{f}$, and proceed to the previous node in $\mathtt{l}$ (if there is none, finish the iteration).

By Invariant (I3), it is clear that in this way we remove from $\mathbb{D}[J]$ exactly all the configurations whose clock values got shifted outside of J, hence Invariants (I2) and (I3) are maintained. As the forest structure was influenced only by removals, Invariants (I4), (I5), and (I6) are maintained as well. Note that the output list $\mathtt{lret}$ is ordered by non-decreasing clock values, as required. As for the time complexity, the procedure presented above takes time $2^{\mathcal{O}(|\mathcal{A}|)} \cdot \ell'$, where ℓ' is the number of nodes that we remove from $\mathtt{l}$. As for every node α the set $\mathtt{states}(\mathtt{root}(\alpha))$ is non-empty and of size at most $|Q|$, with every removed node we add to $\mathtt{lret}$ between 1 and $|Q|$ new configurations. Hence, we can also bound the complexity by $2^{\mathcal{O}(|\mathcal{A}|)} \cdot \ell$, where ℓ is the number of configurations that appear in the output list $\mathtt{lret}$.

Update by a letter. We proceed to the method $\mathtt{updateLetter}(a)$, where $a \in \Sigma$. As argued before, every clock condition appearing in $\mathcal{A}$ is either true for all clock values in J, or false for all clock values in J. For every subset of states $S \subseteq Q$, let $\Phi(S)$ be the set of all states q such that there is a transition $(p, a, q, \gamma, \emptyset)$ in E for some $p \in S$ and clock condition γ that is true in J. In other words, $\Phi(S)$ comprises states reachable from the states of S by non-resetting transitions over a that are available for clock values in J. We define $\Psi(S)$ in a similar way, but for resetting transitions over a that are available for clock values in J.

First, we compute the output of the method, which is $\{(q, 0) : q \in \Psi(S)\}$ where S is the set of all states appearing in the configurations of L. Note that, by Invariant (I4), S can be computed in time $2^{\mathcal{O}(|\mathcal{A}|)}$ by iterating through the list $\mathtt{r}$ and computing the union of sets $\mathtt{states}(\beta)$ for roots β appearing on it. Thus, the output can be computed in time $2^{\mathcal{O}(|\mathcal{A}|)}$.

Second, we need to update the values of function λ by applying all possible non-resetting transitions over a. This can be done by iterating through the list $\texttt{r}$ and, for each root β appearing on it, substituting $\texttt{states}(\beta)$ with $\Phi(\texttt{states}(\beta))$. Note that since we assumed that for every state q, some transition over a is always available at q, it follows that Φ maps non-empty sets of states to non-empty sets of states. Hence, after this substitution the roots of $\texttt{f}$ will still be assigned non-empty sets of states. However, Invariant (I5) may cease to hold, as some roots may now be assigned the same set of states.

We fix this as follows. For every root β of $\texttt{f}$, inspect the list $\texttt{r}$ and find the root β' that has the largest rank among those satisfying $\texttt{states}(\beta) = \texttt{states}(\beta')$. If $\beta = \beta'$, then do nothing. Otherwise, turn β into a non-root node of $\texttt{f}$, remove it from the list $\texttt{r}$, set $\texttt{parent}(\beta) = \beta'$, and increment $\#\texttt{children}(\beta')$ by one. Note that after applying this modification, the function λ stored in the data structure stays the same, while Invariant (I5) becomes satisfied.

As for the other invariants, the satisfaction of Invariants (I2), (I3), and (I4) after the update is clear. However, we need to be careful about Invariant (I6), as we might have substantially modified the structure of the forest $\texttt{f}$. Observe that each modification of $\texttt{f}$ that we applied boils down to attaching a tree with a root of some rank i as a child of a tree with a root of some rank $j > i$. By Invariant (I6), the former tree has depth at most $i + 1$, which is bounded from above by j. Thus, after the attachment, the depth of the latter tree cannot become larger than $j + 1$. We conclude that Invariant (I6) is maintained as well.

Finally, note that since the number of roots of $\texttt{f}$ is always bounded by $2^{|Q|} - 1$, all the operations described above can be performed in time $2^{\mathcal{O}(|\mathcal{A}|)}$.

5 Concluding remarks and future work

In this work we studied the dynamic acceptance problem for timed automata processing data streams. We designed a suitable data structure for one-clock timed automata, where the amortized update time depends only on the size of the automaton. We leave as an open question whether this result can be generalised to the case of multiple clocks.

More generally speaking, it seems that our work identifies dynamic variants of classic automata problems as a potential area of interest for the paradigm of parameterized dynamic data structures. More precisely, if the automaton model in question allows for the device to potentially be in an unbounded number of configurations, then the dynamic maintenance of this set of configurations is a computationally challenging problem, as show-cased in this paper. There are multiple models of devices where similar questions can be asked. Examples include counter automata, register automata, weighted automata, or pushdown automata.

References

1 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014*, pages 434–443. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.53.

2 Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. *SIAM J. Comput.*, 47(3):1098–1122, 2018. doi:10.1137/15M1050987.

3 Josh Alman, Matthias Mnich, and Virginia Vassilevska Williams. Dynamic parameterized problems and algorithms. *ACM Trans. Algorithms*, 16(4):45:1–45:46, 2020. doi:10.1145/3395037.

4 Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994. doi:10.1016/0304-3975(94)90010-8.

5 Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *22nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2002*, pages 1–16, 2002.

6 Ajesh Babu, Nutan Limaye, Jaikumar Radhakrishnan, and Girish Varma. Streaming algorithms for language recognition problems. *Theoretical Computer Science*, 494:13–23, 2013.

7 Max Bannach, Zacharias Heinrich, Rüdiger Reischuk, and Till Tantau. Dynamic kernels for hitting sets and set packing. *Electron. Colloquium Comput. Complex.*, 26:146, 2019. URL: `https://eccc.weizmann.ac.il/report/2019/146`.

8 Béatrice Bérard and Catherine Dufourd. Timed automata and additive clock constraints. *Inf. Process. Lett.*, 75(1-2):1–7, 2000. `doi:10.1016/S0020-0190(00)00075-2`.

9 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering conjunctive queries under updates. In *36th ACM SIGMOD-SIGACT-SIGAI symposium on Principles of database systems*, pages 303–318, 2017.

10 Patricia Bouyer, Samy Jaziri, and Nicolas Markey. Efficient timed diagnosis using automata with timed domains. In *18th International Conference on Runtime Verification, RV 2018*, pages 205–221, 2018. `doi:10.1007/978-3-030-03769-7_12`.

11 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 *2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 796–809. SIAM, 2021. `doi:10.1137/1.9781611976465.50`.

12 Lorenzo Clemente and Sławomir Lasota. Timed pushdown automata revisited. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015*, pages 738–749, 2015. `doi:10.1109/LICS.2015.73`.

13 Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, 31(6):1794–1813, 2002.

14 Moses Ganardi. *Language recognition in the sliding window model*. PhD thesis, Universität Siegen, 2019. `doi:10.25819/ubsi/464`.

15 Moses Ganardi, Danny Hucke, Daniel König, Markus Lohrey, and Konstantinos Mamouras. Automata theory on sliding windows. In *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018*, pages 31:1–31:14, 2018.

16 Moses Ganardi, Danny Hucke, and Markus Lohrey. Querying regular languages over sliding windows. In *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016*, pages 18:1–18:14, 2016.

17 Monika Rauch Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. Computing on data streams. *External memory algorithms*, 50:107–118, 1998.

18 Muhammad Idris, Martín Ugarte, Stijn Vansummeren, Hannes Voigt, and Wolfgang Lehner. Efficient query processing for dynamically changing datasets. *ACM SIGMOD Record*, 48(1):33–40, 2019.

19 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3SUM conjecture. In *27th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016*, pages 1272–1287. SIAM, 2016. `doi:10.1137/1.9781611974331.ch89`.

20 Martin Leucker and Christian Schallhart. A brief account of runtime verification. *J. Log. Algebr. Program.*, 78(5):293–303, 2009. `doi:10.1016/j.jlap.2008.08.004`.

21 Philip M Lewis, Richard Edwin Stearns, and Juris Hartmanis. Memory bounds for recognition of context-free and context-sensitive languages. In *6th Annual Symposium on Switching Circuit Theory and Logical Design, SWCT 1965*, pages 191–202. IEEE, 1965.

22 Frédéric Magniez, Claire Mathieu, and Ashwin Nayak. Recognizing well-parenthesized expressions in the streaming model. *SIAM Journal on Computing*, 43(6):1880–1905, 2014.

23 Mihai Pătraşcu. Towards polynomial lower bounds for dynamic problems. In *42nd ACM Symposium on Theory of Computing, STOC 2010*, pages 603–610. ACM, 2010. `doi:10.1145/1806689.1806772`.

24 Kanat Tangwongsan, Martin Hirzel, and Scott Schneider. Low-latency sliding-window aggregation in worst-case constant time. In *11th ACM International Conference on Distributed and Event-based Systems*, pages 66–77, 2017.

25 Prasanna Thati and Grigore Rosu. Monitoring algorithms for metric temporal logic specifications. *Electron. Notes Theor. Comput. Sci.*, 113:145–162, 2005. doi:10.1016/j.entcs.2004.01.029.

26 Stavros Tripakis. Fault diagnosis for timed automata. In *7th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT 2002*, pages 205–224, 2002. doi:10.1007/3-540-45739-9_14.

Close Relatives (Of Feedback Vertex Set), Revisited

Hugo Jacob
ENS Paris-Saclay, France

Thomas Bellitto
Sorbonne Université, CNRS, LIP6 UMR 7606, Paris, France

Oscar Defrain
Aix-Marseille Université, CNRS, LIS UMR 7020, Marseille, France

Marcin Pilipczuk
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Abstract

At IPEC 2020, Bergougnoux, Bonnet, Brettell, and Kwon (*Close Relatives of Feedback Vertex Set Without Single-Exponential Algorithms Parameterized by Treewidth*, IPEC 2020, LIPIcs vol. 180, pp. 3:1–3:17) showed that a number of problems related to the classic Feedback Vertex Set (FVS) problem do not admit a $2^{o(k \log k)} \cdot n^{\mathcal{O}(1)}$-time algorithm on graphs of treewidth at most k, assuming the Exponential Time Hypothesis. This contrasts with the $3^k \cdot k^{\mathcal{O}(1)} \cdot n$-time algorithm for FVS using the Cut&Count technique.

During their live talk at IPEC 2020, Bergougnoux et al. posed a number of open questions, which we answer in this work.

- Subset Even Cycle Transversal, Subset Odd Cycle Transversal, Subset Feedback Vertex Set can be solved in time $2^{\mathcal{O}(k \log k)} \cdot n$ in graphs of treewidth at most k. This matches a lower bound for Even Cycle Transversal of Bergougnoux et al. and improves the polynomial factor in some of their upper bounds.
- Subset Feedback Vertex Set and Node Multiway Cut can be solved in time $2^{\mathcal{O}(k \log k)} \cdot n$, if the input graph is given as a cliquewidth expression of size n and width k.
- Odd Cycle Transversal can be solved in time $4^k \cdot k^{\mathcal{O}(1)} \cdot n$ if the input graph is given as a cliquewidth expression of size n and width k. Furthermore, the existence of a constant $\varepsilon > 0$ and an algorithm performing this task in time $(4 - \varepsilon)^k \cdot n^{\mathcal{O}(1)}$ would contradict the Strong Exponential Time Hypothesis.

A common theme of the first two algorithmic results is to represent connectivity properties of the current graph in a state of a dynamic programming algorithm as an auxiliary forest with $\mathcal{O}(k)$ nodes. This results in a $2^{\mathcal{O}(k \log k)}$ bound on the number of states for one node of the tree decomposition or cliquewidth expression and allows to compare two states in $k^{\mathcal{O}(1)}$ time, resulting in linear time dependency on the size of the graph or the input cliquewidth expression.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases feedback vertex set, treewidth, cliquewidth

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.21

Related Version *Full Version*: https://arxiv.org/abs/2106.16015

Funding This research is part of a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme Grant Agreement 714704. This research was conducted while Hugo Jacob was doing a research internship at the University of Warsaw.

1 Introduction

Treewidth, introduced by Robertson and Seymour in their seminal Graph Minors series [28], but also independently introduced under different names by other authors, is probably the most successful graph width notion. (For the formal definition of treewidth and other width notions mentioned in this introduction, we refer to Section 2.) From the algorithmic point of view, its applicability is described by Courcelle's theorem [10] that asserts that every problem expressible in monadic second order logic with quantification over vertex sets and edge sets, can be solved in linear time (in the size of the graph) on graphs of bounded treewidth.

Due to the abundance of algorithms for graphs of bounded treewidth, their use in practice, and since Courcelle's theorem provides a very weak bound on the dependency of the running time of the algorithm on the treewidth of the input graph, a lot of research in the last decade has been devoted to understanding optimal running time bounds for algorithms on graphs of bounded treewidth. One of the first methodological approaches was provided by two works of Lokshtanov, Marx, and Saurabh [22, 23, 24, 25]. Their contribution can be summarized as follows.

- For a number of classic problems, the known (and very natural) dynamic programming algorithm, given an n-vertex graph G and a tree decomposition of width k, runs in time $c^k \cdot n^{\mathcal{O}(1)}$ for a constant $c > 1$. [22, 24] shows that in most cases the constant c is optimal, assuming the Strong Exponential Time Hypothesis.[1]
- [23, 25] introduces a framework for proving lower bounds (assuming the Exponential Time Hypothesis) against $2^{o(k \log k)} \cdot n^{\mathcal{O}(1)}$-time algorithms with the same input as above.

Both aforementioned works seemed to point to a general conclusion that the natural and naive dynamic programming algorithms on graphs of bounded treewidth are probably optimal in essentially all interesting cases. This intuition has been refuted by Cygan et al. [16] who presented the Cut&Count technique which allowed $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$-time algorithms on graphs of treewidth k for many connectivity problems where the natural and naive algorithm runs in time $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(1)}$. One of the prominent examples of such problems is Feedback Vertex Set (FVS) where, given a graph G and an integer p, one asks for a set of at most p vertices that hits all cycles of G.

Since then, the intricate landscape of optimal algorithms parameterized by the treewidth has been explored by many authors, see e.g. [1, 2, 3, 4, 7, 8, 14, 15, 27, 29]. Last year at IPEC 2020, Bergougnoux, Bonnet, Brettell, and Kwon [5] presented an in-depth study of problems related to FVS, showing that for most of them $2^{\mathcal{O}(k \log k)}$ is the optimal (assuming ETH) dependency on treewidth in the running time bound. During their live talk at IPEC 2020, they asked a number of open questions. In this work, we continue this line of research and answer all of them.

Hitting cycles in graphs of bounded treewidth

We first focus on the problems Odd Cycle Transversal (OCT) and Even Cycle Transversal (ECT) where, given a graph G and an integer p, the goal is to pick a set of at most p vertices of G that hits all odd cycles (resp. even cycles) of G. These problems are thus closely related to the aforementioned FVS problem that asks to hit *all* cycles. Using the fact that graphs without odd cycles are exactly bipartite graphs, it is relatively easy to obtain a $3^k \cdot k^{\mathcal{O}(1)} \cdot n$-time algorithm for OCT for graphs equipped with a tree decomposition of width k [17], and the base 3 of the exponent is optimal assuming SETH [22, 24].

[1] For a discussion on the complexity assumptions used, namely the Exponential Time Hypothesis (ETH) and the Strong Exponential Time Hypothesis (SETH), we refer to Chapter 14 of [13].

In contrast to FVS and OCT, Bergougnoux et al. [5] showed that, assuming ETH, ECT admits no $2^{o(k \log k)} \cdot n^{\mathcal{O}(1)}$-time algorithm and asked for a matching upper bound. Our first result is a positive answer to this question, even in a more general setting of Subset Even Cycle Transversal (SECT). In Subset Feedback Vertex Set (SFVS), Subset Odd Cycle Transversal (SOCT), and Subset Even Cycle Transversal, given a graph G, a set $S \subseteq V(G)$, and an integer p, the goal is to find a set of at most p vertices that hits every cycle (resp. odd cycle, even cycle) that passes through a vertex of S.

▶ **Theorem 1.** *Subset Feedback Vertex Set, Subset Odd Cycle Transversal and Subset Even Cycle Transversal, even in the weighted setting, can be solved in time* $2^{\mathcal{O}(k \log k)} \cdot n$ *on* n*-vertex graphs of treewidth* k*.*

Here, and in later statements, by *weighted setting* we mean the following: every vertex has its positive integer weight, and the input integer p becomes an upper bound on the total weight of the solution.

Misra, Raman, Ramanujan, and Saurabh [26] showed that a graph G does not contain an even cycle if and only if every block (2-connected component) of G is an edge or an odd cycle. The key ingredient of the proof of Theorem 1 for SECT is a characterization (in the same spirit, but more involved) of graphs G with sets $S \subseteq V(G)$ that do not contain an even cycle passing through a vertex of S.

This improves the polynomial factor of the running time bound of [5] for SOCT and SFVS from cubic to linear.

Clique-width parameterization

We then switch our attention to clique-width. Clique-width is a width measure aiming at capturing simple yet (contrary to treewidth) dense graphs. It originates from works of Courcelle, Engelfriet, and Rozenberg [11] and of Wanke [30] from early 90s. Informally speaking, a graph G is of clique-width at most k if one can provide an expression (called a *k-expression*) that constructs G using only k *labels* which essentially are names for vertex sets. Clique-width plays the role of treewidth for dense graphs in the following sense: any problem expressible in monadic second order logic with quantification over vertex sets (but not edge sets) can be solved in time $f(k) \cdot n$, given a k-expression of size n constructing the input graph, where f is some computable function [12]. Similarly as for treewidth, it is natural to investigate optimal functions f in such running time bounds. Here, the most relevant works are due to Bui-Xuan, Suchý, Telle, and Vatshelle [9] who showed an algorithm with $f(k) = 2^{\mathcal{O}(k \log k)}$ for FVS, and Bergougnoux and Kanté [6] who later showed an algorithm with $f(k) = 2^{\mathcal{O}(k)}$ by adapting the algorithm of Bodlaender et al. [7] for graphs of bounded treewidth to the context of bounded clique-width.

One should also mention a long line of work [18, 19, 20] searching for optimal running time bounds on graphs of bounded clique-width for problems *not* captured by the aforementioned meta-theorem and that provably (unless FPT = W[1]) do not have algorithms with the running time bound $f(k) \cdot n^{\mathcal{O}(1)}$, given a k-expression building the input graph.

Following on the open questions provided by Bergougnoux et al., we focus on SFVS and Node Multiway Cut (NMwC). In the second problem, given a graph G, a set $T \subseteq V(G)$, and an integer p, the goal is to find a set of at most p vertices that does not contain any vertex of T, but hits all paths with both endpoints in T. We show the following.

▶ **Theorem 2.** *Subset Feedback Vertex Set and Node Multiway Cut, even in the weighted setting, can be solved in time* $2^{\mathcal{O}(k \log k)} \cdot n$ *if the input graph is given as a* k*-expression of size* n*.*

Note that the running time bound of Theorem 2 matches the lower bound of Bergougnoux et al. [5] for pathwidth parameterization[2] of SFVS and NMwC, and it is straightforward to turn a path decomposition of width ℓ into a k-expression for $k = \ell + \mathcal{O}(1)$.

Observe also that, if vertex weights are allowed, NMwC reduces to SFVS. Namely, given a NMwC instance (G, T, p), set the weights of all vertices of T to $+\infty$, create a graph G' by adding to G a new vertex s of weight $+\infty$ adjacent to all vertices of T and set $S := \{s\}$; the SFVS instance (G', S, p) is easily seen to be equivalent to the input NMwC instance (G, T, P). Since it is straightforward to turn a k-expression of G into a $(2k)$-expression of G', in Theorem 2 it suffices to focus only on the SFVS problem.

A common theme in the dynamic programming algorithm of Theorem 1 and of Theorem 2 is the representation of the connectivity in the currently analyzed graph as an auxiliary forest of size $\mathcal{O}(k)$ with some annotations. This allows a neat description of the essential connectivity features, avoiding involved case analysis. The $\mathcal{O}(k)$ bound serves two purposes. First, it implies a bound of $2^{\mathcal{O}(k \log k)}$ on the number of states of the dynamic programming algorithm at one node of the tree decomposition or k-expression. Second, it allows to perform computations on states in $k^{\mathcal{O}(1)}$ time, giving the final linear dependency on the size of the graph or the input k-expression in the running time bound.

Hitting odd cycles in graphs of bounded clique-width

Finally, we restrict our attention to Odd Cycle Transversal. Recall that in graphs of treewidth k, OCT admits an algorithm with running time bound $3^k \cdot k^{\mathcal{O}(1)} \cdot n$ [17] and the base 3 is optimal assuming SETH [22, 24]. We show that for clique-width, the optimal base is 4.

▶ **Theorem 3.** *Odd Cycle Transversal, even in the weighted setting, can be solved in time $4^k \cdot k^{\mathcal{O}(1)} \cdot n$ if the input graph is given as a k-expression of size n. Furthermore, the existence of a constant $\varepsilon > 0$ and an algorithm performing the same task in time $(4-\varepsilon)^k \cdot n^{\mathcal{O}(1)}$ contradicts the Strong Exponential Time Hypothesis.*

The key insight in the OCT algorithm of [17] is to reformulate the problem into finding explicitly a partition $V(G) = X \uplus A \uplus B$ that minimizes $|X|$ while keeping $G[A]$ and $G[B]$ both edgeless. Then, in a dynamic programming algorithm on a tree decomposition, one remembers the assignment of the vertices of the current bag into X, A, and B; this yields the 3^k factor in the time complexity. For clique-width, a similar approach yields 4^k states: every label may be allowed to contain only vertices of X, allowed to contain vertices of X or A but not B, allowed to contain vertices of X or B but not A, or allowed to contain vertices of any of the three sets. To obtain the upper bound of Theorem 3, one needs to add on top of the above an appropriate convolution-like treatment of the disjoint union nodes of the k-expression. The lower bound of Theorem 3 combines a way to encode evaluation of two variables of a CNF-SAT formula into one of the four aforementioned states of a single label with a few gadgets for checking in the OCT regime if a clause is satisfied, borrowed from the corresponding reduction for pathwidth from [22, 24].

This extended abstract contains only an overview of the proofs of Theorems 1 and 2.

[2] We do not formally define pathwidth in this work, as it is not used except for this paragraph.

2 Preliminaries

For most standard definitions and notations used in this paper, we refer to the preliminaries section in the full version of the paper. We recall here only more nonstandard definitions.

A *rooted tree* is a tree T together with a special vertex $r \in V(T)$, called the *root*. It induces a natural ancestor-descendent relation $\leq$ on its vertex set, where a vertex $s \in V(T)$ is said to be a *descendent* of a vertex $t \in V(T)$, denoted $s \leq t$, if t is on the (unique) path from s to r in T.

To capture the parity of lengths of paths in a robust manner, we use graphs with edges labeled with elements of $\mathbb{F}_2$. Let G be a graph where every edge $e \in E(G)$ is assigned an element $\lambda(e) \in \mathbb{F}_2$. With a walk W in G we can associate then the sum of the elements assigned to the edges on W (with multiplicities, i.e., if an edge e appears c times in W, then we add $c \cdot \lambda(e)$ to the sum). An important observation is that if in G in every closed walk the edge labels sum up to 0, then for every $u, v \in V(G)$, in every walk from u to v the edge labels sum up to the same value, depending only on u and v. Furthermore, one can in linear time (a) check if every closed walk in G sums up to 0 and, if this is the case, (b) compute for every u a value $x_u \in \mathbb{F}_2$, called henceforth the *potential*, such that for every $u, v \in V(G)$ and every walk W from u to v in G, the sum of the labels of W equals $x_v - x_u$. Indeed, it suffices to take any rooted spanning forest F of G, define x_u to be the sum of the labels on the path from u to the root of the corresponding tree in F, and check for every $uv \in E(G)$ if $\lambda(uv) = x_v - x_u$.

▶ **Definition 4** (Nice tree decomposition). *A nice tree decomposition of a graph G is a rooted tree decomposition $(T, \{X_t\}_{t \in V(T)})$ such that:*

- *the root and leaves of T have empty bags; and*
- *other nodes are of one of the following types:*
  - ***Introduce vertex node**: a node t with exactly one child t' such that $X_t = X_{t'} \cup \{v\}$ with $v \notin X_{t'}$. We say that v is introduced at t;*
  - ***Forget vertex node**: a node t with only one child t' such that $X_t = X_{t'} \setminus \{v\}$ with $v \in X_{t'}$. We say that v is forgotten at t; and*
  - ***Join node**: a node t with two children t_1, t_2 such that $X_t = X_{t_1} = X_{t_2}$.*

For each node t of the decomposition, we define a partial *graph $G_t = G\left[\bigcup_{s \leq t} X_s\right] - E(G[X_t])$.*

Note that edges of partial graphs appear at forget vertex nodes and that they correspond to adding edges between the forgotten vertex and its neighbours.

From a tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ of G of width k, a nice tree decomposition of width k with $\mathcal{O}(k|V(G)|)$ nodes can be computed in time $\mathcal{O}(k^2 \cdot \max(|V(T)|, |V(G)|))$, see [21].

A *k-labeled* graph is a graph G together with a labeling function $\Gamma : V(G) \to [k]$. For k-labeled graphs H, G, and integers $i, j \in [k]$, we consider the following operations:

- **Vertex creation:** $i(v)$ is the k-labeled graph consisting of a single vertex v with label i;
- **Disjoint union:** $H \oplus G$ is the k-labeled graph consisting of the disjoint union of H and G;
- **Join:** $\eta_{i \times j}(G)$ is the k-labeled graph obtained by adding an edge between any pair of vertices one being of label i, the other of label j, if the edge does not exist; and
- **Renaming label:** $\rho_{i \to j}(G)$ is the k-labeled graph obtained by changing the label of every vertex labeled i to label j: $\forall v \in \Gamma^{-1}(\{i\}),\ \Gamma(v) := j$

The *clique-width* of a graph G, denoted $\mathbf{cw}(G)$, is the least integer k such that a k-labeled graph isomorphic to G can be constructed using these operations. We call *k-expression* of a k-labeled graph G a sequence of operations that leads to the construction of G. Note that such a sequence defines a tree, called *tree associated to the k-expression* in the following.

Consider a k-expression of a k-labeled graph G, and its associated tree T. For a node $t \in V(T)$, we denote by T_t the subtree of T rooted at t, and associate it with the labeled graph G_t it describes. For an integer $i \in [k]$, we denote by $V_i(G)$ the set of vertices of label i in G. By an abuse of notations in the following, by "label i" for a labeled graph G we may refer to both the integer i, or the set $V_i(G)$.

We define *partially k-labeled graphs* as labeled graphs with a labeling function $\Gamma(G) : V(G) \to [k] \cup \{\bot\}$ and call *unlabeled* the vertices of $\Gamma^{-1}(\{\bot\})$.

Given a graph G and a set of vertices $S \subseteq V(G)$, we call S-vertex a vertex that is part of S and we call S-path (resp. S-cycle) a path that contains at least one S-vertex.

When a 2-connected multigraph contains a cycle, we call it *nontrivial*. Other 2-connected multigraphs are the degenerate cases of a single vertex and a bridge, i.e., two vertices connected by a single edge. A *nontrivial* 2-connected component of a multigraph is a 2-connected component which is a nontrivial 2-connected multigraph, it is not an isolated vertex or a bridge.

Since our algorithms solve weighted variants of the problems, we will denote by $c : V(G) \to \mathbb{Z} \cup \{+\infty\}$ the weight function of the instance. We extend this notation to sets of vertices with $c(U) = \sum_{v \in U} c(v)$. The unweighted variant corresponds to having $c(v) = 1$ for all $v \in V(G)$.

In the context of a dynamic programming algorithm, a state is a tuple of parameters used to index the table in which computations are done. We denote our table by d. We call transition from a set of states $\mathcal{A}$ to a single state B the action of updating the entry indexed by B based on the values of states in $\mathcal{A}$. Since we consider only minimizing problems, for a function f, such a transition will consist in applying the operation

$$d[B] := \min\{d[B], f(\mathcal{A})\}.$$

We denote this operation by $d[B] \leftarrow f(\mathcal{A})$ and say that value $f(\mathcal{A})$ is propagated to state B.

3 Hitting even cycles in graphs of bounded treewidth

In this section we highlight the main ideas behind Theorem 1. Rather than simply giving an algorithm for just SOCT and SECT, we also show how our method gives less involved algorithms for SFVS and ECT. All these problems can be seen as looking for a minimum deletion set such that the resulting graph has no odd S-cycle, no even S-cycle, no S-cycle, and no even cycle. In order to have a common notation, we will call $\square$-cycles the cycles that have to be hit in the problem and $\square$-cycle-free the graphs that do not contain $\square$-cycles.

To transform a $\square$-cycle-free graph into a forest, we will replace its nontrivial 2-connected components with tree structures. We use labeled vertices to store efficiently the properties of these nontrivial 2-connected components.

We begin by giving a characterisation of nontrivial 2-connected $\square$-cycle-free graphs for each problem. This implies characterisations of $\square$-cycle-free graphs.

▶ **Lemma 5.** *Let G be a nontrivial 2-connected multigraph.*

1. *G contains no S-cycle if and only if it contains no S-vertex.*
2. *G contains no even cycle if and only if it is an odd cycle.*
3. *G contains no odd S-cycle if and only if it has one of the following forms:*
   - *G contains no S-vertex and is not bipartite*
   - *G contains no S-vertex and is bipartite*
   - *G contains at least one S-vertex and is bipartite.*

4. *G contains no even S-cycle if and only if it has of one of the following forms:*
   - *G contains no S-vertex and is not bipartite*
   - *G contains no S-vertex and is bipartite*
   - *G contains at least one S-vertex, the connected components of $G - S$ are bipartite, together with S-vertices they form a cycle: each S-vertex has degree 2 and each connected component of $G - S$ has degree 2. One S-cycle is odd. We later call* bipartite subcomponents *the connected components of $G - S$. This is illustrated in Figure 1a.*

The first point is immediate, the second follows from [26, Lemma 1], and the third was observed in [5, Lemma 13]. The last point was not known to us and we provide a proof.

Proof. Suppose that G is a nontrivial 2-connected multigraph containing no even S-cycle.

If G contains no S-vertex, it is either bipartite or not, leading to the first possible forms.

If G contains an S-vertex, it must contain an S-cycle C due to being a nontrivial 2-connected multigraph and C is odd because G contains no even S-cycle.

▷ Claim 6. If two vertices are connected by three disjoints paths at least two of which are S-paths then two of the paths form an even S-cycle.

Proof. The three cycles formed by combining the paths are S-cycles and they cannot all be odd: if we denote them C_1, C_2, C_3, $|C_3| = |C_1| + |C_2| - 2|C_1 \cap C_2|$. ◁

Consider a connected component A of $G - V(C)$.

Consider an S-vertex v of A, because G is a nontrivial 2-connected multigraph, there exist two disjoint paths that connect v to distinct vertices a, b of C, a and b satisfy the conditions of claim 6 leading to a contradiction. Hence, A cannot contain an S-vertex.

Since G is a nontrivial 2-connected multigraph, there are at least 2 edges between A and distinct vertices of C. Consider 2 arbitrary distinct such edges, they cut C into two paths P_1 and P_2 with extremities u and v. Since A is connected, there is a third u–v path P_3 through A. Only one of P_1 and P_2 may contain an S-vertex, by claim 6 applied to P_1, P_2, P_3. In particular, u and v cannot be S-vertices, this implies that the only edges incident to S-vertices in G are edges of cycle C, so S-vertices have degree 2.

Let $\widetilde{A}$ be the connected component of $G - S$ containing A. $\widetilde{A}$ contains a maximal S-free path of C because A is connected to C and cannot be adjacent to S-vertices. $\widetilde{A}$ contains only one maximal S-free path of C because otherwise either we get two edges from A to C that separate C in two S-paths and this was excluded in the previous paragraph, or we have a chord ab in C that connects two distinct maximal S-free-paths and this is excluded by Claim 6. In particular, note that this shows that $\widetilde{A}$ has outdegree 2 in G.

Consider a cycle C' of $\widetilde{A}$, then there are 2 disjoint paths from it to the S-vertices adjacent to $\widetilde{A}$. If they are distinct we can connect them with a disjoint path via C. This constructions contains two S-cycles C and $C \Delta C'$ which must both be odd so C' can only be even. Hence $\widetilde{A}$ contains no odd cycle so it is bipartite.

We can conclude that all connected components of $G - S$ are bipartite and that together with S-vertices they form a cycle.

Conversely, if G contains no S-vertex it does not contain any even S-cycle. If it is a cycle of bipartite components and S-vertices with one S-cycle C being odd, then each S-cycle C' goes through all bipartite components and S-vertices. Replacing the path of C by the path of C' in each bipartite component preserves parity because endpoints are unique. We conclude that all S-cycles are odd in G. ◀

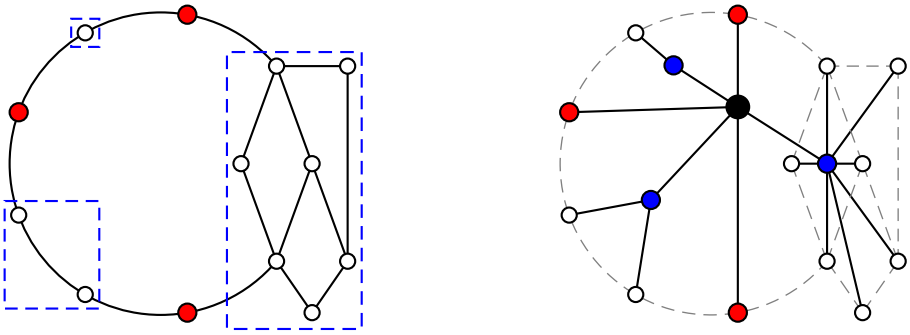
▶ **Definition 7.** *Given a $\square$-cycle-free graph G, we define its* underlying forest $F(G)$ *as the graph obtained from G by modifying independently each nontrivial 2-connected component C as follows:*

SFVS. *Remove edges inside C and add an unlabeled vertex adjacent to all vertices of C.*

ECT. *Remove edges inside C and add a vertex adjacent to all vertices of C and label it "odd cycle".*

SOCT. *Remove edges inside C and add a vertex adjacent to all vertices of C, label it "bipartite" or "not bipartite" based on the property of C and make it an S-vertex if C contains an S-vertex.*

SECT. *In the two first forms we remove edges inside C and add a vertex adjacent to all vertices of C and label it "bipartite" or "not bipartite" based on the property of C. For the last form, for each bipartite subcomponent of C, we remove its edges, add a vertex labeled "internal bipartite" adjacent to its vertices. Then remove edges of C incident to S, add an S-vertex labeled "odd cycle" adjacent to S-vertices and vertices labeled "internal bipartite". This is illustrated in Figure 1b.*



(a) An example of graph with no even S-cycle. The vertices of S are depicted in red. The blue boxes denote the bipartite subcomponents.

(b) The underlying forest we build from the graph on Figure 1a. The "internal bipartite" vertices are depicted in blue and the "odd cycle" vertex is black.

Figure 1 The last form of SECT: "internal bipartite" vertices.

Observe that, because labeled vertices are only introduced by this underlying forest, to each labeled vertex v, we can associate a nontrivial 2-connected component C: the one that resulted in the creation of v. Observe also that for a path P between two unlabeled vertices, if it contains a labeled vertex, then it contains a vertex of its associated component before it on P and another vertex of its associated component after it on P.

We now introduce reduction rules that allow us to maintain a simplified description of underlying forests relatively to a given subset of vertices, that we call *active*. Vertices that are not active are called *inactive*. Theses rules and this terminology largely resemble what is done in [5].

▶ **Definition 8.** *Given a $\square$-cycle-free graph G, its underlying forest $F(G)$ and subset of active vertices X, a reduced underlying forest F_r is obtained by applying exhaustively the following rules on $F(G)$:*

- *Delete inactive vertices of degree at most one.*
- *For each maximal path P with internal inactive vertices of degree 2, we replace it with a path P' with same endpoints, such that P' contains exactly one occurrence of each label present in P and a single S-vertex if P contained one, where endpoints are considered to be contained in P and P'.*

For SECT we add another rule: if a maximal path with internal inactive vertices of degree 2 contains at least 2 vertices labeled "internal bipartite" but no vertex labeled "odd cycle", we keep 2 occurences of the label "internal bipartite".

The set of reduced underlying forests obtained from $F(G)$ with active vertices X is denoted $F(G, X)$.

Observe that a reduced forest is not unique, however properties that we will show on them will not depend on the choice of representative.

Standard arguments show that a reduced forest has bounded size.

▶ **Lemma 9.** *In a problem using K label symbols (including S-membership), $F \in F(G, X)$ has at most $(K+1)(2|X|-2)+1$ vertices.*

The crucial property preserved by a reduced forest is the following.

▶ **Lemma 10.** *For $F \in F(G, X)$, for each pair of active vertices u and v, there is a path between them in G if and only if there is a path between them in F. For each type of nontrivial 2-connected component, a u–v path in G goes through at least one such component if and only if the u–v path in F contains a vertex with the corresponding label symbol ("internal bipartite" counts for the bipartite subcomponent but also the S-cycle containing it). There exists a u–v path in G containing an S-vertex if and only if there exists a u–v path in F containing an S-vertex or a vertex labeled "internal bipartite". If there is a u–v path in F, every unlabeled vertex that is on the u–v path in F is also on all u–v paths in G.*

A property that is not preserved by a reduced forest is the length of paths. Since we are only interested in parity, we maintain a $\mathbb{F}_2$-labeling α of edges. We say that α is a *valid* $\mathbb{F}_2$-labeling of $F \in F(G, X)$ if, there exists β a $\mathbb{F}_2$-labeling of the edges of $F(G)$ such that edges incident to vertices labeled "bipartite" or "internal bipartite" are labeled 0 for one side of the bipartition and 1 for the other side, edges incident to other labeled vertices are labeled 0, and edges between unlabeled vertices are labeled 1, and for each edge uv of F, its label is the sum of labels on the edges of the u–v path in $F(G)$. During the application of reduction rules, each edge is given as its label the sum of labels of the path that was connecting its endpoints.

▶ **Lemma 11.** *For $F \in F(G, X)$, for each pair of active vertices u and v connected in G, all u–v paths in G have same parity if and only if the path between u and v in F contains no vertex with label symbol "odd cycle", "not bipartite" or "internal bipartite". Furthermore, when this condition is satisfied, the parity of the paths in G is given by the sum of labels on the edges of F.*

The main technical engine of our algorithms is the following join operation.

▶ **Lemma 12.** *There exists a polynomial-time algorithm that, for every pair of $\square$-cycle-free graphs G_1 and G_2 with $V(G_1) \cap V(G_2) = X$, given on input two reduced forests with valid $\mathbb{F}_2$-labelings (F_1, α_1) and (F_2, α_2), with $F_1 \in F(G_1, X)$ and $F_2 \in F(G_2, X)$, decides whether $G_1 \cup G_2$ is $\square$-cycle-free and, in case of a positive answer, computes a reduced forest $F \in F(G_1 \cup G_2, X)$ and, except for the SFVS problem, a valid $\mathbb{F}_2$-labeling α.*

With Lemma 12 in hand, assembling a dynamic programming algorithm of Theorem 1 is a tedious but straightforward exercise. The proof of Lemma 12 requires a careful consideration of all possible 2-connected components the graph $F_1 \cup F_2$ may contain and how to treat them in the considered problems.

4 Subset Feedback Vertex Set in graphs of bounded cliquewidth

We describe a dynamic programming algorithm to solve Subset Feedback Vertex Set on clique-width expressions. With a bottom-up computation, it builds small labeled forests that describe the graphs that can be obtained by vertex deletion.

A state of our dynamic programming will consist of a node of the k-expression, a partially labeled forest, and a label state assignment $\mathcal{P} : [k] \to \mathcal{Q}$, with $\mathcal{Q} = \{Q_\varnothing, Q_1, Q_1^*, Q_2, Q_w, Q_w^*, Q_f\}$ the set of label states. State $Q_\varnothing$ is assigned to labels that are completely contained in the current deletion set. States Q_1 and Q_1^* are assigned to labels consisting of a single non-S-vertex, or a single S-vertex, respectively. States Q_w and Q_w^* are called *waiting states*: they are assigned to labels for which we have guessed that they will be joined (only once) to a non-S-vertex from a label in state Q_1, or to an S-vertex from a label in state Q_1^*, respectively. State Q_2 is assigned to labels having at least two vertices not in S: it is assigned to labels for which we have guessed that they will be joined (potentially several times) to either a vertex from a label in state Q_1, or to vertices from a label in state Q_2. These guessing tricks can be seen as a form of what is called "expectation from the outside" in [9]. We point that guessing these joins implies that labels in states Q_w, Q_w^*, Q_2 will eventually be connected – this is detailed below. At last, state Q_f is called *final state*: it will contain vertices that will not be joined anymore, and hence that may be unlabeled. To summarize, states in $\mathcal{Q}$ express the following constraints on joins:

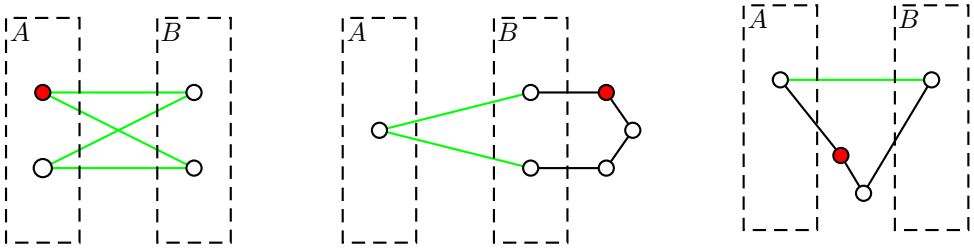
- joins with a label in state $Q_\varnothing$ will be ignored;
- no join with a label in state Q_f will be performed;
- labels in state Q_w (resp. Q_w^*) will only be joined with those in state Q_1 (resp. Q_1^*); and
- labels in state Q_2 will never be joined with those in state Q_1^*.

Now, considering an S-cycle-free graph $\widetilde{G}$ obtained by vertex deletion, we will say that a label i is *compatible* with label state:

- $Q_\varnothing$ if no vertex of $\widetilde{G}$ is labeled i;
- Q_1 if exactly one vertex of $\widetilde{G}$ is labeled i, and it is not in S;
- Q_1^* if exactly one vertex of $\widetilde{G}$ is labeled i, and it is in S;
- Q_2 if at least two vertices of $\widetilde{G}$ are labeled i, they are not in S, and no S-path in $\widetilde{G}$ has both its endpoints labeled i;
- Q_w if at least two vertices of $\widetilde{G}$ are labeled i, at least one S-vertex is labeled i, and no S-path in $\widetilde{G}$ has both its endpoints labeled i;
- Q_w^* if at least two vertices of $\widetilde{G}$ are labeled i, no path in $\widetilde{G}$ has both its endpoints labeled i; and
- Q_f if at least two vertices of $\widetilde{G}$ are labeled i.

These conditions, together with the constraints on joins that are expressed above, aim to capture cases for which a join between labels of pairs of label states will not create S-cycles – this will be explicited in proofs and illustrated in Figure 2. In the following, we say that a label state assignment $\mathcal{P}$ is *compatible* with $\widetilde{G}$ if each label is compatible with its state in this graph. Note that looking at the properties of vertices in a label in part gives the label state assignment that it should have: the conflicts are for choosing between Q_f, Q_w^* and, based on the presence or not of an S-vertex, either Q_w or Q_2. This is expected because these states contain the information on a guess on what will later be added to the graph.

Let us now introduce an auxiliary partially labeled graph which will conveniently represent the connectedness implied by guesses we made so far when assigning labels to label states, while simplifying the manipulation of labels. We point that this auxiliary graph will *not* be computed by the algorithm: it shall only be used in the proofs. Given a labeled graph $\widetilde{G}$



(a) Case 1: $|A| \geq 2$, $|B| \geq 2$, and $|A \cap S| + |B \cap S| \geq 1$.

(b) Case 2: $|A| \geq 1$, $|B| \geq 2$, and there is an S-path with endpoints in B.

(c) Case 3: $|A| \geq 1$, $|B| \geq 1$, and there is an S-path connecting a vertex of A to one of B.

Figure 2 The three cases when a join (depicted in green) creates an S-cycle. The figures illustrates the smallest number of vertices of S required. Thus, up to symmetry, the vertices depicted in red have to be in S while the vertices in white may or may not be in S.

and a label state assignment $\mathcal{P}$, we denote by $H(\widetilde{G}, \mathcal{P})$ the partially labeled graph obtained from $\widetilde{G}$, by conducting the following modifications for each label i:

- if i is in state Q_2 or Q_w, we add a vertex labeled i, connect it to other vertices labeled i, and unlabel these vertices, making the added vertex the only vertex labeled i;
- if i is in state Q_w^*, we add an S-vertex labeled i, connect it to other vertices labeled i, and unlabel these vertices, making the added vertex the only vertex labeled i; and
- if i is in state Q_f, we unlabel vertices labeled i.

Note that in the auxiliary graph, we add vertices that are not part of the original graph. The role of these vertices – for states Q_2, Q_w, and Q_w^* – is to represent the label i as if it was connected (which will eventually be the case as we guessed a later join), as well as manipulating nonempty labels as single vertices: for $\widetilde{G}$ compatible with $\mathcal{P}$, each nonempty label i contains exactly one vertex in $H(\widetilde{G}, \mathcal{P})$, which we call *representative* of i in $H(\widetilde{G}, \mathcal{P})$, and that we denote by $h(i)$.

Recall that, when $\mathcal{P}$ is compatible with $\widetilde{G}$, some connectedness conditions are satisfied by label states. We say that a partially labeled multigraph $\widehat{F}$ *expresses the connectedness* in $H(\widetilde{G}, \mathcal{P})$, for $\widetilde{G}$ and $\mathcal{P}$ compatible, if:

- for each label i, there is at most one vertex labeled i in $\widehat{F}$;
- to every vertex $h(i)$ in $H(\widetilde{G}, \mathcal{P})$ corresponds a vertex $r(i)$ labeled i in $\widehat{F}$: we call it the *representative* of label i in $\widehat{F}$, and $r(i)$ is an S-vertex if and only if $h(i)$ is an S-vertex; and
- for any two vertices $h(i), h(j)$ in $H(\widetilde{G}, \mathcal{P})$, there exists a $h(i)$–$h(j)$ path in $H(\widetilde{G}, \mathcal{P})$ if and only if there exists a $r(i)$–$r(j)$ path in $\widehat{F}$, and there exists a $h(i)$–$h(j)$ S-path in $H(\widetilde{G}, \mathcal{P})$ if and only if there exists a $r(i)$–$r(j)$ S-path in $\widehat{F}$.

We are now ready to introduce reduction rules which, when applied on the multigraph $\widehat{F}$ expressing the connectedness in $H(\widetilde{G}, \mathcal{P})$, will produce the aforementioned partially labeled forest. The idea behind this forest is that, to check the existence of (S-)paths linking representatives of labels i and j, unlabeled vertices of degree at most two in such (S-)paths may be "contracted" as long as we do not remove all (S-)vertices on these paths. In the following for a partially labeled multigraph $\widehat{F}$, we denote by $\mathrm{Red}(\widehat{F})$ the forest obtained from $\widehat{F}$ by applying the following reduction rules:

- for each nontrivial 2-connected component C, we introduce an unlabeled vertex, call it *central vertex* of C, connect it to vertices of C, and remove all other edges inside the component;

- we iteratively remove unlabeled vertices of degree at most one;
- for each maximal S-path with internal unlabeled vertices of degree two, we replace it by connecting the endpoints to a single new unlabeled S-vertex; and
- for each maximal path with internal unlabeled vertices of degree two that is not an S-path, we replace it by a single edge between its endpoints.

It is easily seen that the produced graph is indeed a forest as the graph of nontrivial 2-connected components of any graph is a tree, and each nontrivial 2-connected component is replaced by a star. Furthermore, reducing F preserves the fact that it expresses the connectedness in $H(\widetilde{G}, \mathcal{P})$: the formal statement is omitted here. As in Lemma 9, we can bound the size of the reduced forest:

▷ Claim 13. $\mathrm{Red}(\widehat{F})$ has $\mathcal{O}(k)$ vertices.

A *state* of the dynamic programming algorithm is a tuple $(t, F, \mathcal{P})$, where $t \in V(T)$, F is a partially labeled forest, and $\mathcal{P} : [k] \to \mathcal{Q}$ is a label state assignment. We say that $(t, F, \mathcal{P})$ is *admissible* if there exists $X \subseteq V(G)$ such that $\mathcal{P}$ is compatible with $G_t - X$, $H(G_t - X, \mathcal{P})$ is S-cycle-free, and F expresses the connectedness in $H(G_t - X, \mathcal{P})$. Our dynamic programming algorithm will not consider all possible states, but compute a value $d[t, F, \mathcal{P}]$ for some states $(t, F, \mathcal{P})$. We call *reachable* a state that is considered by the algorithm. We will show that reachable states are admissible, that for every $t \in V(T)$, for each $X \subseteq V(G_t)$, if $G_t - X$ is S-cycle-free, then there exists a reachable state $(t, F, \mathcal{P})$ such that $d[t, F, \mathcal{P}] \leq |X|$, and that the optimal value for SFVS on the given instance is the minimum of values $d[r, F, \mathcal{P}]$ where r is the root of the k-expression.

First, let us slightly modify our clique-width expression in order to simplify the description of our computations. We double the set of labels, denoting them by $\{1, ..., k, 1', ..., k'\}$, and replace each disjoint union node t with children t_1, t_2 by the following subexpression: $\rho_{1' \to 1}(\dots \rho_{k' \to k}(G_{t_1} \oplus (\rho_{1 \to 1'}(\dots \rho_{k \to k'}(G_{t_2})))))$. This gives the property that in disjoint union nodes, each label is used by at most one of the children nodes.

We now describe the bottom-up computation of reachable states for each possible type of node in the clique-width expression.

Leaf node. If t is a leaf node with $G_t = i(v)$, two cases arise. Either v is deleted which is described by state $(t, F_\varnothing, \mathcal{P}_\varnothing)$ initialized with value $c(v)$, where $F_\varnothing$ is the empty graph, and $\mathcal{P}_\varnothing$ is the function that maps every $i \in [k]$ to $Q_\varnothing$. Otherwise we keep v, which is described by state $(t, F, \mathcal{P})$ where F consists of the isolated vertex v, $\mathcal{P}(i) = Q_1^*$ if $v \in S$, $\mathcal{P}(i) = Q_1$ otherwise, and, for all $j \neq i$, $\mathcal{P}(j) = Q_\varnothing$.

Join node. Let t be a join node with $G_t = \eta_{i \times j}(G_{t'})$. For each reachable state $(t', F', \mathcal{P}')$, we proceed as follows. If the representatives of i and j are connected by an S-path in F', we do nothing. Otherwise, we will construct states $(t, F, \mathcal{P})$ defined in the following cases, depending on $\mathcal{P}'(i)$ and $\mathcal{P}'(j)$, starting with $F := F'$ and $\mathcal{P} := \mathcal{P}'$:

- if one of i and j is in state $Q_\varnothing$, we do not modify F nor $\mathcal{P}$;
- if i and j are in states Q_1 or Q_1^*, we add an edge between the representatives of i and j in F;
- if i and j are in states Q_1 or Q_2, we add an edge between the representatives of i and j in F, and if i or j are in state Q_2 they are allowed to change to Q_f in $\mathcal{P}$, if they do we also unlabel their representative: we enumerate all possibilities here;
- if i and j are in states Q_1^* and Q_w^*, we identify their representative in F: the resulting vertex has its label in state Q_1^*, and the label in state Q_w^* is assigned state Q_f in $\mathcal{P}$; and
- if i and j are in states Q_1 and Q_w, we identify their representative in F: the resulting vertex has its label in state Q_1, and the label in state Q_w is assigned state Q_f in $\mathcal{P}$.

For each such cases, we reduce F and propagate the value $d[t', F', \mathcal{P}']$ to the states $(t, F, \mathcal{P})$, where $\mathcal{P}$ is the modified label state assignment.

Renaming label node. Let t be a renaming label node with $G_t = \rho_{i\to j}(G_{t'})$. For each reachable state $(t', F', \mathcal{P}')$, we construct states $(t, F, \mathcal{P})$ starting with $\mathcal{P} := \mathcal{P}$ and $F := F'$ by first setting $\mathcal{P}(i) = Q_\varnothing$, and proceeding as follows depending on $\mathcal{P}'(i)$ and $\mathcal{P}'(j)$:

- if i and j are in a state among $\{Q_f, Q_1, Q_1^*\}$, we unlabel the representatives of i and j in F', and set $\mathcal{P}(j) = Q_f$;
- if one of i and j is in state $Q_\varnothing$, then either i is in state $Q_\varnothing$ and we do nothing, or j is in state $Q_\varnothing$, we assign it to the other label state, and the vertex of F labeled i is relabeled j;
- if i and j are in state Q_1, and the representatives of i and j are not connected by a path in F', in F, we add an S-vertex labeled j, connect it to these vertices, and unlabel them. Label j is then assigned state Q_w^* in $\mathcal{P}$;
- if one of i and j is in state Q_1^*, the other is in state Q_1 or Q_1^*, and the representatives of i and j are not connected by a path in F', we consider two possibilities depending on whether they will be joined to a vertex of S, or to a vertex of $V(G) \setminus S$. First, in F, we add a new vertex labeled j, connect it to the representatives of i and j, and unlabel the representatives of i and j. Then, if the new vertex is chosen to be in S, j is assigned state Q_w^* in $\mathcal{P}$. Otherwise, j is assigned state Q_w in $\mathcal{P}$;
- if i and j are in states Q_α and Q_β, for $\alpha, \beta \in \{1, 2, w\}$, and the representatives of i and j are not connected by an S-path in F', in F, we identify the representatives of i and j: the resulting vertex is of label j, and j is assigned state Q_δ in $\mathcal{P}$ with $\delta := \max_{1<2<w} \{2, \alpha, \beta\}$;
- if one of i and j is in state Q_1^*, the other is in state Q_w or Q_2, and the representatives of i and j are not connected by a path in F', in F, we add an edge between the representatives of i and j, and the representative of the label in state Q_1^* becomes unlabeled, while the other vertex is given label j. Label j is assigned state Q_w in $\mathcal{P}$; and
- if one of i and j is in state Q_w^*, the other is in state Q_1 or Q_1^*, and the representatives of i and j are not connected by a path in F', in F, we add an edge between the representatives of i and j, and the representative of the label in state Q_1 or Q_1^* becomes unlabeled, while the other vertex is given label j. Label j is assigned state Q_w^* in $\mathcal{P}$.

For each such cases, we reduce F, and we propagate the value $d[t', F', \mathcal{P}']$ to the state $(t, F, \mathcal{P})$, where $\mathcal{P}$ is the modified label state assignment.

Disjoint union node. If t is a disjoint union node with $G_t = G_{t_1} \oplus G_{t_2}$, for each pair of reachable states $(t_1, F_1, \mathcal{P}_1)$, $(t_2, F_2, \mathcal{P}_2)$, since they use disjoint sets of labels, we can simply define $F = F_1 \oplus F_2$. The label state assignment $\mathcal{P}$ is defined by $\mathcal{P}(i) = \mathcal{P}_1(i)$ for $i \in [k]$ and $\mathcal{P}(i') = \mathcal{P}_2(i')$ for $i' \in \{1', \ldots, k'\}$. The value $d[t_1, F_1, \mathcal{P}_1] + d[t_2, F_2, \mathcal{P}_2]$ is propagated to state $(t, F, \mathcal{P})$.

We conclude the section noting that, with the above described transitions, proving the correctness of the algorithm by induction is a tedious, but rather straightforward exercise. It basically consists on considering all the different label states in which labels i and j may lie when performing join and relabel operations, together with the compatibility of the current S-cycle-free graph and label states assignments.

References

1 Julien Baste, Ignasi Sau, and Dimitrios M. Thilikos. Hitting minors on bounded treewidth graphs. IV. an optimal algorithm. *CoRR*, abs/1907.04442, 2019. arXiv:1907.04442.

2 Julien Baste, Ignasi Sau, and Dimitrios M. Thilikos. Hitting minors on bounded treewidth graphs. I. general upper bounds. *SIAM J. Discret. Math.*, 34(3):1623–1648, 2020. doi:10.1137/19M1287146.

3 Julien Baste, Ignasi Sau, and Dimitrios M. Thilikos. Hitting minors on bounded treewidth graphs. II. single-exponential algorithms. *Theor. Comput. Sci.*, 814:135–152, 2020. doi:10.1016/j.tcs.2020.01.026.

4 Julien Baste, Ignasi Sau, and Dimitrios M. Thilikos. Hitting minors on bounded treewidth graphs. III. lower bounds. *J. Comput. Syst. Sci.*, 109:56–77, 2020. doi:10.1016/j.jcss.2019.11.002.

5 Benjamin Bergougnoux, Édouard Bonnet, Nick Brettell, and O-joung Kwon. Close Relatives of Feedback Vertex Set Without Single-Exponential Algorithms Parameterized by Treewidth. In Yixin Cao and Marcin Pilipczuk, editors, *15th International Symposium on Parameterized and Exact Computation (IPEC 2020)*, volume 180 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 3:1–3:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.IPEC.2020.3.

6 Benjamin Bergougnoux and Mamadou Moustapha Kanté. Fast exact algorithms for some connectivity problems parameterized by clique-width. *Theor. Comput. Sci.*, 782:30–53, 2019. doi:10.1016/j.tcs.2019.02.030.

7 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Inf. Comput.*, 243:86–111, 2015. doi:10.1016/j.ic.2014.12.008.

8 Marthe Bonamy, Lukasz Kowalik, Jesper Nederlof, Michal Pilipczuk, Arkadiusz Socala, and Marcin Wrochna. On directed feedback vertex set parameterized by treewidth. In Andreas Brandstädt, Ekkehard Köhler, and Klaus Meer, editors, *Graph-Theoretic Concepts in Computer Science - 44th International Workshop, WG 2018, Cottbus, Germany, June 27-29, 2018, Proceedings*, volume 11159 of *Lecture Notes in Computer Science*, pages 65–78. Springer, 2018. doi:10.1007/978-3-030-00256-5_6.

9 Binh-Minh Bui-Xuan, Ondrej Suchý, Jan Arne Telle, and Martin Vatshelle. Feedback vertex set on graphs of low clique-width. *Eur. J. Comb.*, 34(3):666–679, 2013. doi:10.1016/j.ejc.2012.07.023.

10 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

11 Bruno Courcelle, Joost Engelfriet, and Grzegorz Rozenberg. Handle-rewriting hypergraph grammars. *J. Comput. Syst. Sci.*, 46(2):218–270, 1993. doi:10.1016/0022-0000(93)90004-G.

12 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.

13 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5. Springer, 2015.

14 Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast hamiltonicity checking via bases of perfect matchings. *J. ACM*, 65(3):12:1–12:46, 2018. doi:10.1145/3148227.

15 Marek Cygan, Dániel Marx, Marcin Pilipczuk, and Michal Pilipczuk. Hitting forbidden subgraphs in graphs of bounded treewidth. *Inf. Comput.*, 256:62–82, 2017. doi:10.1016/j.ic.2017.04.009.

16 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on*

Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011, pages 150–159. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.23.

17 Samuel Fiorini, Nadia Hardy, Bruce A. Reed, and Adrian Vetta. Planar graph bipartization in linear time. *Discret. Appl. Math.*, 156(7):1175–1180, 2008. doi:10.1016/j.dam.2007.08.013.

18 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of clique-width parameterizations. *SIAM J. Comput.*, 39(5):1941–1956, 2010. doi:10.1137/080742270.

19 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Almost optimal lower bounds for problems parameterized by clique-width. *SIAM J. Comput.*, 43(5):1541–1563, 2014. doi:10.1137/130910932.

20 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Clique-width III: hamiltonian cycle and the odd case of graph coloring. *ACM Trans. Algorithms*, 15(1):9:1–9:27, 2019. doi:10.1145/3280824.

21 Ton Kloks. *Treewidth: computations and approximations*, volume 842. Springer Science & Business Media, 1994.

22 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 777–789. SIAM, 2011.

23 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Slightly superexponential parameterized problems. In Dana Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 760–776. SIAM, 2011. doi:10.1137/1.9781611973082.60.

24 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. *ACM Trans. Algorithms*, 14(2):13:1–13:30, 2018. doi:10.1145/3170442.

25 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Slightly superexponential parameterized problems. *SIAM J. Comput.*, 47(3):675–702, 2018. doi:10.1137/16M1104834.

26 Pranabendu Misra, Venkatesh Raman, MS Ramanujan, and Saket Saurabh. Parameterized algorithms for even cycle transversal. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 172–183. Springer, 2012.

27 Michal Pilipczuk. Problems parameterized by treewidth tractable in single exponential time: A logical approach. In Filip Murlak and Piotr Sankowski, editors, *Mathematical Foundations of Computer Science 2011 - 36th International Symposium, MFCS 2011, Warsaw, Poland, August 22-26, 2011. Proceedings*, volume 6907 of *Lecture Notes in Computer Science*, pages 520–531. Springer, 2011. doi:10.1007/978-3-642-22993-0_47.

28 Neil Robertson and Paul D. Seymour. Graph minors. III. planar tree-width. *J. Comb. Theory, Ser. B*, 36(1):49–64, 1984. doi:10.1016/0095-8956(84)90013-3.

29 Ignasi Sau and Uéverton dos Santos Souza. Hitting forbidden induced subgraphs on bounded treewidth graphs. In Javier Esparza and Daniel Král', editors, *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic*, volume 170 of *LIPIcs*, pages 82:1–82:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.MFCS.2020.82.

30 Egon Wanke. k-nlc graphs and polynomial algorithms. *Discret. Appl. Math.*, 54(2-3):251–266, 1994. doi:10.1016/0166-218X(94)90026-4.

Long Paths Make Pattern-Counting Hard, and Deep Trees Make It Harder

Vít Jelínek
Computer Science Institute, Charles University, Prague, Czech Republic

Michal Opler
Computer Science Institute, Charles University, Prague, Czech Republic

Jakub Pekárek
Department of Applied Mathematics, Charles University, Prague, Czech Republic

Abstract

We study the counting problem known as #PPM, whose input is a pair of permutations π and τ (called *pattern* and *text*, respectively), and the task is to find the number of subsequences of τ that have the same relative order as π. A simple brute-force approach solves #PPM for a pattern of length k and a text of length n in time $O(n^{k+1})$, while Berendsohn, Kozma and Marx have recently shown that under the exponential time hypothesis (ETH), it cannot be solved in time $f(k)n^{o(k/\log k)}$ for any function f. In this paper, we consider the restriction of #PPM, known as $\mathcal{C}$-Pattern #PPM, where the pattern π must belong to a hereditary permutation class $\mathcal{C}$. Our goal is to identify the structural properties of $\mathcal{C}$ that determine the complexity of $\mathcal{C}$-Pattern #PPM.

We focus on two such structural properties, known as the *long path property* (LPP) and the *deep tree property* (DTP). Assuming ETH, we obtain these results:

1. If $\mathcal{C}$ has the LPP, then $\mathcal{C}$-Pattern #PPM cannot be solved in time $f(k)n^{o(\sqrt{k})}$ for any function f, and
2. if $\mathcal{C}$ has the DTP, then $\mathcal{C}$-Pattern #PPM cannot be solved in time $f(k)n^{o(k/\log^2 k)}$ for any function f.

Furthermore, when $\mathcal{C}$ is one of the so-called monotone grid classes, we show that if $\mathcal{C}$ has the LPP but not the DTP, then $\mathcal{C}$-Pattern #PPM can be solved in time $f(k)n^{O(\sqrt{k})}$. In particular, the lower bounds above are tight up to the polylog terms in the exponents.

2012 ACM Subject Classification Mathematics of computing → Permutations and combinations; Theory of computation → Pattern matching; Theory of computation → Problems, reductions and completeness

Keywords and phrases Permutation pattern matching, subexponential algorithm, conditional lower bounds, tree-width

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.22

Related Version *Full Version*: `https://arxiv.org/abs/2111.03479`

Funding *Vít Jelínek*: Supported by project 18-19158S of the Czech Science Foundation.
Michal Opler: Supported by project 21-32817S of the Czech Science Foundation and by project SVV–2020–260578.

1 Introduction

One of the most frequently studied algorithmic problems related to permutations is known as Permutation pattern matching (or PPM). The input of PPM is a pair of permutations τ (the "text") of length n and π (the "pattern") of length k, and the goal is to determine whether τ contains π as a subpermutation (see Section 2 for formal definitions).

In full generality, PPM is NP-complete, as shown by Bose et al. [4]. Thus most research into PPM focuses either on improved exact algorithms, or on identifying special types of inputs for which the PPM can be solved in polynomial time, or at least in subexponential time. Note that a direct brute-force approach solves PPM in time $O(n^{k+1})$.

A particularly fruitful technique to solving PPM has been proposed by Ahal and Rabinovich [1], who showed that PPM can be solved in time $n^{O(\mathrm{tw}(\pi))}$, where $\mathrm{tw}(\pi)$ denotes the tree-width of the so-called *incidence graph* of the pattern π. The bound was subsequently tightened to $n^{\mathrm{tw}(\pi)+1}$ by Berendsohn, Kozma and Marx [3], who have used it to show that PPM can be solved in time $n^{k/4+o(k)}$.

Another approach to PPM, due to Guillemot and Marx [9] (with a slight improvement by Fox [8]) shows that the problem can be solved in time $n \cdot 2^{O(k^2)}$, implying that the problem is fixed-parameter tractable with parameter k.

Closely related to PPM is its counting version #PPM, whose goal is to compute the number of occurrences of the pattern π in the text τ. Berendsohn et al. [3] show that their bounds of $O(n^{\mathrm{tw}(\pi)+1})$ and $n^{k/4+o(k)}$ for PPM also apply to solving #PPM. In contrast, the FPT result for PPM by Guillemot and Marx [9] likely does not extend to #PPM, since Berendsohn et al. [3] show that, under the exponential time hypothesis (ETH), #PPM cannot be solved in time $f(k)n^{o(k/\log k)}$, for any function f.

Given that both PPM and #PPM are hard in general, it is natural to consider their complexity on restricted inputs. A common approach is to fix a hereditary class $\mathcal{C}$ of permutations, and study the restriction of PPM or #PPM to inputs where the pattern π belongs to $\mathcal{C}$. Such restriction is known as $\mathcal{C}$-Pattern PPM and $\mathcal{C}$-Pattern #PPM, respectively. It follows from the results of Ahal and Rabinovich [1] and Berendsohn et al. [3], that the restricted problems are polynomial whenever the function $\mathrm{tw}(\pi)$ is bounded on the class $\mathcal{C}$. This idea is the basis for previous results establishing sharp thresholds between polynomial and NP-hard cases of $\mathcal{C}$-Pattern PPM [11, 12]. In fact, in all the known cases when $\mathcal{C}$-Pattern PPM and $\mathcal{C}$-Pattern #PPM are polynomial, the class $\mathcal{C}$ has bounded tree-width.

While distinguishing the polynomial cases of $\mathcal{C}$-Pattern PPM from the NP-hard ones is obviously the main focus of research, it is also of interest to distinguish subexponential cases from those cases which (under suitable complexity assumptions, such as the ETH) require exponential or near-exponential time. Here again, the tree-width plays a key role. It is convenient to associate to a class $\mathcal{C}$ its *tree-width growth function*

$$\mathrm{tw}_{\mathcal{C}}(k) = \max\{\mathrm{tw}(\pi);\ \pi \in \mathcal{C} \wedge |\pi| = k\}.$$

Indeed, Berendsohn et al. [3], extending previous results by Guillemot and Vialette [10], have shown that when $\mathcal{C}$ is the class of 2-monotone permutations (i.e., the permutations merged from two monotone sequences), then $\mathrm{tw}_{\mathcal{C}}(k) = O(\sqrt{k})$, and consequently $\mathcal{C}$-Pattern #PPM can be solved in the subexponential time $n^{O(\sqrt{k})}$. They show, however, that for the class of 3-monotone permutations, the tree-width growth is of order $\Omega(k/\log k)$. Later Berendsohn [2, Theorem 4.1] showed that for the class $\mathcal{C} = \mathrm{Av}(654321)$, consisting of permutations that can be merged from 5 increasing subsequences, $\mathcal{C}$-Pattern #PPM cannot be solved in time $f(k)n^{o(k/\log^4 k)}$ for any function f, unless ETH fails.

In the context of $\mathcal{C}$-Pattern PPM and $\mathcal{C}$-Pattern #PPM, most of the research focuses on the cases when $\mathcal{C}$ is a *principal class*, i.e., the class $\mathrm{Av}(\sigma)$ of all the permutations that avoid a single forbidden pattern σ. Unfortunately, principal classes seldom admit a suitable structural characterisation of their elements, and even in those cases where such characterisations exist, they are very different from one class to another. This makes it hard to obtain general results that apply uniformly to a large set of principal classes.

To sidestep this issue, we mostly avoid dealing with individual principal classes directly, and instead we primarily focus on a different type of permutation classes, the so-called monotone grid classes. We then consider two structural properties of a general permutation

class $\mathcal{C}$, called the *long path property* (LPP) and the *deep tree property* (DTP). Both these properties can be viewed as stating that $\mathcal{C}$ contains monotone grid subclasses of a particular type. We establish lower bounds for the complexity of $\mathcal{C}$-Pattern #PPM applicable to any class $\mathcal{C}$ with LPP or DTP. The definitions of LPP and DTP are somewhat technical (see Section 3); however, it is usually not too hard to verify whether a given class has these properties. Indeed, we are able to identify all the principal classes that have LPP, as well as all those that have DTP; see Subsection 3.2.

The LPP has already played a central part in a dichotomy result of the authors [12], and implicitly also in the work of Berendsohn [2] and Berendsohn et al. [3]. These previous results imply that for a monotone grid class $\mathcal{C}$ these properties are equivalent (assuming $P \neq NP$): (i) $\mathcal{C}$ has LPP, (ii) $\mathrm{tw}_{\mathcal{C}}(k)$ is unbounded, (iii) $\mathrm{tw}_{\mathcal{C}}(k) = \Omega(\sqrt{k})$, and (iv) $\mathcal{C}$-Pattern PPM is NP-complete. For all we know, the equivalence might hold for an arbitrary hereditary class $\mathcal{C}$, i.e., not just a monotone grid class. However, we do not even know whether every class of unbounded tree-width has LPP.

The DTP is a strengthening of LPP, which we introduce in this paper, with the aim of distinguishing the cases of $\mathcal{C}$-Pattern #PPM that can be solved in the subexponential time $f(k)n^{O(\sqrt{k})}$ from those that cannot be solved in time $f(k)n^{o(k/\log k)}$. While LPP forces tree-width growth of order $\Omega(\sqrt{k})$, DTP forces tree-width growth of order $\Omega(k/\log k)$.

Our main results show that the lower bounds on tree-width imposed by LPP and DTP are accompanied by the corresponding complexity lower bounds for $\mathcal{C}$-Pattern #PPM. Specifically, we show that under ETH, the following holds for any permutation class $\mathcal{C}$ (see Theorem 18):

- If $\mathcal{C}$ has the LPP, then $\mathcal{C}$-Pattern #PPM cannot be solved in time $f(k)n^{o(\sqrt{k})}$ for any function f, and
- if $\mathcal{C}$ has the DTP, then $\mathcal{C}$-Pattern #PPM cannot be solved in time $f(k)n^{o(k/\log^2 k)}$ for any function f.

In addition, we show that for classes with LPP, the Ahal–Rabinovich PPM algorithm with complexity $n^{O(\mathrm{tw}(\pi))}$ is asymptotically optimal. More precisely, we show that if ETH holds, then for a class $\mathcal{C}$ with LPP, no algorithm may solve $\mathcal{C}$-Pattern PPM in time $f(t)n^{o(t)}$ for any function f, where $t = \mathrm{tw}(\pi)$ (see Theorem 15). All these complexity lower-bounds are presented in Section 4.

Recall that by a result of Berendsohn et al. [3], the class $\mathcal{C} = \mathrm{Av}(321)$ has tree-width growth $\mathrm{tw}_{\mathcal{C}}(k) = O(\sqrt{k})$, and therefore $\mathcal{C}$-Pattern #PPM can be solved in time $n^{O(\sqrt{k})}$. It turns out that this class has LPP, which implies, by our results above, that $\mathrm{tw}_{\mathcal{C}}(k) = \Omega(\sqrt{k})$ and that $\mathcal{C}$-Pattern #PPM cannot be solved in time $f(k)n^{o(\sqrt{k})}$ for any function f. In particular, both the tree-width bound and the complexity bound are tight.

For any class $\mathcal{C}$ with DTP, the tree-width lower bound $\Omega(k/\log k)$ and the complexity lower-bound $f(k)n^{o(k/\log^2 k)}$ both match, up to the logarithmic terms, the trivial upper bounds of k and $n^{O(k)}$, respectively.

As we mentioned before, we mostly focus on monotone grid classes. We will show that for a monotone grid class $\mathcal{C}$, both LPP and DTP can be easily characterised in terms of a certain graph associated to a monotone grid class $\mathcal{C}$, called the cell graph, and that these two properties asymptotically determine $\mathrm{tw}_{\mathcal{C}}(\cdot)$. An earlier paper of the authors [12] shows that a monotone grid class has bounded tree-width (and hence neither LPP nor DTP) if and only if its cell graph is acyclic. We extend this result as follows (see Corollary 7):

- If the cell graph of a monotone grid class $\mathcal{C}$ is not acyclic but has at most one cycle in each component, then $\mathcal{C}$ has LPP but not DTP, and $\mathrm{tw}_{\mathcal{C}}(k) \in \Theta(\sqrt{k})$.
- If the cell graph of a monotone grid class $\mathcal{C}$ has a component with at least two cycles, then $\mathcal{C}$ has DTP and $\mathrm{tw}_{\mathcal{C}}(k) \in \Omega(k/\log k)$.

2 Preliminaries

A *permutation of length* n is a sequence in which each element of the set $[n] = \{1, 2, \ldots, n\}$ appears exactly once. When writing out short permutations explicitly, we shall omit all punctuation and write, e.g., 15342 for the permutation $1, 5, 3, 4, 2$. The *permutation diagram* of π is the set of points $S_\pi = \{(i, \pi_i);\ i \in [n]\}$ in the plane. Observe that no two points from S_π share the same x- or y-coordinate. We say that such a set is in *general position*.

For a point p in the plane, we denote its horizontal coordinate as $p.x$, and its vertical coordinate as $p.y$. Two finite sets $S, R \subseteq \mathbb{R}^2$ in general position are *isomorphic* if there is a bijection $f\colon S \to R$ such that for any pair of points $p \neq q$ of S we have $f(p).x < f(q).x$ if and only if $p.x < q.x$, and $f(p).y < f(p).y$ if and only if $p.y < q.y$. The *reduction* of a finite set $S \subseteq \mathbb{R}^2$ in general position is the unique permutation π such that S is isomorphic to S_π. We write $\pi = \mathrm{red}(S)$.

We say that a permutation τ *contains* a permutation π, written $\pi \leq \tau$, if the diagram of τ contains a subset that is isomorphic to the diagram of π. If τ does not contain π, we say that it *avoids* π. A *permutation class* is a set $\mathcal{C}$ of permutations which is *hereditary*, i.e., for every $\sigma \in \mathcal{C}$ and every $\pi \leq \sigma$, we have $\pi \in \mathcal{C}$. For a permutation π, we let $\mathrm{Av}(\pi)$ denote the set of all the permutations that avoid π; this is clearly a permutation class. The class $\mathrm{Av}(21)$ of all the increasing permutations and the class $\mathrm{Av}(12)$ of all the decreasing permutations are denoted by the symbols $\boxslash$ and $\boxbslash$, respectively.

We will frequently refer to symmetries that transform permutations into other permutations. For our purposes, it is convenient to describe these symmetries geometrically, as transformations of the plane acting on permutation diagrams. We define the *m-box* to be the set $(\frac{1}{2}, m + \frac{1}{2}) \times (\frac{1}{2}, m + \frac{1}{2})$. Observe that for every permutation π of length at most m, the permutation diagram S_π is a subset of the m-box. We view permutation symmetries as bijections acting of the whole m-box. There are eight such symmetries, generated by:

reversal which reflects the m-box horizontally, i.e. the image of point p is $(m + 1 - p.x, p.y)$,

complement which reflects the m-box vertically, i.e. the image of point p is $(p.x, m+1-p.y)$,

inverse which reflects the m-box through its main diagonal, i.e. the image of point p is $(p.y, p.x)$.

In particular, the reversal of a permutation $\pi = \pi_1, \ldots, \pi_n$ is the permutation $\pi^r = \pi_n \pi_{n-1}, \ldots, \pi_1$, the complement of π is the permutation $\pi^c = n + 1 - \pi_1, n + 1 - \pi_2, \ldots, n + 1 - \pi_n$, and the inverse π^{-1} is the permutation $\sigma = \sigma_1, \ldots, \sigma_n$ such that $\sigma_i = j \iff \pi_j = i$. We also apply these symmetries to sets of permutations, in an obvious way: if Ψ is one of the eight symmetries defined above and $\mathcal{C}$ is a permutation class, we define $\Psi(\mathcal{C})$ as $\{\Psi(\pi);\ \pi \in \mathcal{C}\}$.

The incidence graph G_π of a permutation $\pi = \pi_1, \ldots, \pi_n$ is the graph whose vertices are the n entries $\pi_1, \ldots, \pi_n$, with two entries π_i and π_j connected by an edge if $|i - j| = 1$ or $|\pi_i - \pi_j| = 1$. In particular, the graph G_π is a union of two paths, one of them visiting the entries of π in left-to-right order, and the other in top-to-bottom order. We let $\mathrm{tw}(\pi)$ denote the tree-width of G_π.

Monotone grid classes

An important type of permutation classes are the so-called monotone grid-classes, which we now define. A *gridding matrix of size $k \times \ell$* is a matrix $\mathcal{M}$ with k columns and ℓ rows, whose every entry is a permutation class. A *monotone gridding matrix* is a gridding matrix whose every entry is one of the three classes $\emptyset$, $\boxslash$ or $\boxbslash$. Note that to be consistent with the Cartesian coordinates that we use to describe permutation diagrams, we will number the

$$\mathcal{M} = \begin{pmatrix} \mathrm{Av}(12) & \longrightarrow & \mathrm{Av}(21) \\ & & \downarrow \\ \mathrm{Av}(21) & - & \mathrm{Av}(12) \end{pmatrix}$$

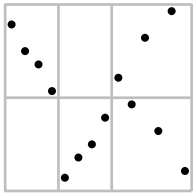


Figure 1 A monotone gridding matrix $\mathcal{M}$ on the left and a permutation equipped with an $\mathcal{M}$-gridding on the right. Empty entries of $\mathcal{M}$ are omitted and the edges of $G_{\mathcal{M}}$ are drawn in $\mathcal{M}$.

rows of a matrix from bottom to top, and we give the column coordinate as the first one. In particular, $\mathcal{M}_{i,j}$ denotes the entry in column i and row j of the matrix $\mathcal{M}$, with $1 \le i \le k$ and $1 \le j \le \ell$.

Let π be a permutation of length n. A $(k \times \ell)$-*gridding* of π is a pair of weakly increasing sequences $1 = c_1 \le c_2 \le \cdots \le c_{k+1} = n+1$ and $1 = r_1 \le r_2 \le \cdots \le r_{\ell+1} = n+1$. For $i \in [k]$ and $j \in [\ell]$, the (i,j)-*cell* of the gridding of π is the set of points $p \in S_\pi$ satisfying $c_i \le p.x < c_{i+1}$ and $r_j \le p.y < r_{j+1}$. Note that each point of the diagram S_π belongs to a unique cell of the gridding. A permutation π together with a gridding (c, r) forms a *gridded permutation*.

Let $\mathcal{M}$ be a gridding matrix of size $k \times \ell$. We say that the gridding of π is an $\mathcal{M}$-*gridding* if for every $i \in [k]$ and $j \in [\ell]$, the subpermutation of π induced by the points in the (i,j)-cell of the gridding of π belongs to the class $\mathcal{M}_{i,j}$.

We let $\mathrm{Grid}(\mathcal{M})$ denote the set of permutations that admit an $\mathcal{M}$-gridding. This is clearly a permutation class. A *monotone grid class* is any permutation class $\mathrm{Grid}(\mathcal{M})$ for a monotone gridding matrix $\mathcal{M}$.

The *cell graph* of a gridding matrix $\mathcal{M}$, denoted $G_{\mathcal{M}}$, is the graph whose vertices are all the pairs (i,j) for which $\mathcal{M}_{i,j}$ is an infinite permutation class. Two vertices are adjacent if they appear in the same row or the same column of $\mathcal{M}$, and there is no other cell containing an infinite class between them. See Figure 1. A *proper-turning path* in $G_{\mathcal{M}}$ is a path P such that no three vertices of P share the same row or column.

Grid transforms and orientations

Let π be a permutation of length n with a $(k \times \ell)$-gridding (c, r), where $c = (c_1, \ldots, c_{k+1})$ and $r = (r_1, \ldots, r_{\ell+1})$. The *reversal of the i-th column* of π is the operation that transforms π into a new permutation π' by taking the rectangle $[c_i, c_{i+1} - 1] \times [1, n]$ and flipping it along its vertical axis, thus producing the diagram of a new permutation π'. Equivalently, π' is created from π by reversing the order of the entries of π at positions $c_i, c_i + 1, \ldots, c_{i+1} - 1$. We view π' as a gridded permutation, with the same gridding (c, r) as π.

Similarly, the *complementation of the j-th row* transforms the diagram of π by flipping the rectangle $[1, n] \times [r_j, r_{j+1} - 1]$ along its horizontal axis, producing the diagram of a new gridded permutation π'.

We may similarly apply reversals to the columns of a gridding matrix $\mathcal{M}$ and complements to its rows. Reversing the i-th column of $\mathcal{M}$ produces a new gridding matrix, in which all the classes in the i-th column of $\mathcal{M}$ are replaced by their reversals. Row complementation of a gridding matrix is defined analogously. Note that a column reversal or a row complementation in a gridded permutation or in a gridding matrix is an involution, i.e., repeating the same operation twice restores the original permutation or matrix. Note also that when we perform a sequence of column reversals and row complementations, then the end result does not depend on the order in which the operations were performed.

To describe succinctly a sequence of row and column operations, we introduce the notion of $(k \times \ell)$-*orientation*, which is a pair of functions $\mathcal{F} = (f_c, f_r)$ with $f_c \colon [k] \to \{-1, 1\}$ and $f_r \colon [\ell] \to \{-1, 1\}$. *Applying* the orientation $\mathcal{F}$ to a $(k \times \ell)$-gridded permutation π produces a new gridded permutation $\mathcal{F}(\pi)$ with the same gridding as π, obtained by reversing each column i such that $f_c(i) = -1$ and complementing each row j such that $f_r(j) = -1$. The application of $\mathcal{F}$ to a gridding matrix $\mathcal{M}$ is defined analogously, and produces a gridding matrix denoted $\mathcal{F}(\mathcal{M})$. Note that (c, r) is an $\mathcal{M}$-gridding of π if and only if it is an $\mathcal{F}(\mathcal{M})$-gridding of $\mathcal{F}(\pi)$.

An orientation $\mathcal{F}$ is a *consistent orientation* of a monotone gridding matrix $\mathcal{M}$, if every nonempty entry of $\mathcal{F}(\mathcal{M})$ is equal to $\boxslash$. As an example, the matrix $\begin{pmatrix} \boxbslash & \boxslash \\ \boxslash & \boxbslash \end{pmatrix}$ has a consistent orientation acting by reversing the first column and complementing the first row. On the other hand, the matrix $\begin{pmatrix} \boxbslash & \boxslash \\ \boxslash & \boxslash \end{pmatrix}$ has no consistent orientation, since applying any orientation to this matrix yields a matrix with an odd number of $\boxbslash$-entries.

The following lemma, due to Vatter and Waton [15], will be later useful.

▶ **Lemma 1.** *Every monotone gridding matrix whose cell graph is acyclic has a consistent orientation.*

Tile assembly

In the hardness reductions that we are about to present, we frequently need to construct permutations whose diagrams have a natural $k \times \ell$ grid-like structure. We describe such a diagram by taking each cell individually and describing the points inside it. For such a description, it is often convenient to assume that each cell has its own coordinate system whose origin is near the bottom-left corner of the cell. This allows us to describe the coordinates of the points inside the cell without referring to the position of the cell within the whole permutation diagram. In effect, we describe the diagram of the gridded permutation by first constructing a set of independent "tiles" $\mathcal{T}_{i,j}$ for $i \in [k]$ and $j \in [\ell]$ of the same size, and then translating each tile $\mathcal{T}_{i,j}$ to column i and row j of the diagram. On top of that, we often need to apply an orientation to the gridded permutation whose diagram we constructed.

We now describe the whole procedure more formally. Fix an integer m and recall that an m-box is a square of the form $(\frac{1}{2}, m + \frac{1}{2}) \times (\frac{1}{2}, m + \frac{1}{2})$. An m-tile is a finite set of points inside the m-box. Note that the coordinates of the points in the tile may not be integers. A $(k \times \ell)$-family of m-tiles is a collection $(\mathcal{T}_{i,j};\ i \in [k], j \in [\ell])$ where each $\mathcal{T}_{i,j}$ is an M-tile. Let $\mathcal{F}$ be a $(k \times \ell)$-orientation. The *$\mathcal{F}$-assembly* of the family $(\mathcal{T}_{i,j};\ i \in [k], j \in [\ell])$ is the gridded permutation obtained as follows.

First, we translate each tile $\mathcal{T}_{i,j}$ by adding $m(i-1)$ to each horizontal coordinate and $m(j-1)$ to each vertical coordinate. Thus, the m-tiles will be disjoint. If the union of the translated tiles is not in general position, we rotate it slightly clockwise to reach general position. Notice that we can do so without changing the relative position of any pair of points that were already in general position. This yields a point set isomorphic to a unique permutation π. See Figure 2. Additionally, π has a natural gridding whose cells correspond to the translated tiles. To finish the construction, we apply the orientation $\mathcal{F}$ to π, obtaining the gridded permutation $\mathcal{F}(\pi)$, which is the $\mathcal{F}$-assembly of the family of tiles $(\mathcal{T}_{i,j};\ i \in [k], j \in [\ell])$.

▶ **Observation 2.** *Let $(\mathcal{T}_{i,j};\ i \in [k], j \in [\ell])$ be a family of tiles, let $\mathcal{F}$ be an orientation, and let $\mathcal{M}$ be a gridding matrix such that $\mathcal{T}_{i,j}$ is isomorphic to a permutation from the class $\mathcal{M}_{i,j}$. Then the $\mathcal{F}$-assembly of the family of tiles $(\mathcal{T}_{i,j};\ i \in [k], j \in [\ell])$ is a permutation from the class $Grid(\mathcal{F}(\mathcal{M}))$.*

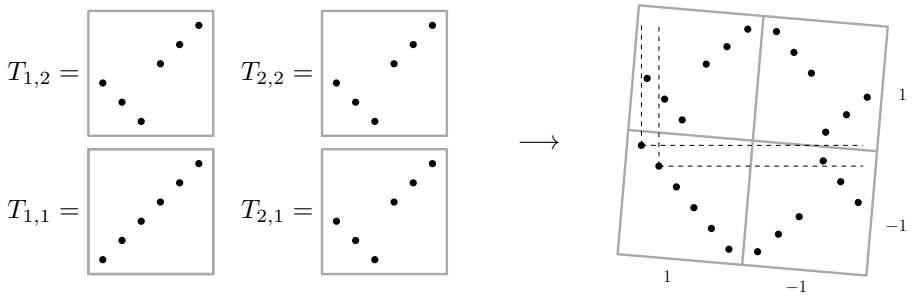


Figure 2 A 2×2 family of tiles $\mathcal{T}$ on the left and its $\mathcal{F}$-assembly on the right for a 2×2 orientation $\mathcal{F}$ given next to each row and column on the right. General position is attained by rotating the resulting point set clockwise. The dashed lines indicate relative positions of two particular points.

3 Tree-width bounds

3.1 Width of monotone grid classes

We say that a permutation class $\mathcal{C}$ has the *long path property* (LPP) if for every k the class $\mathcal{C}$ contains a monotone grid subclass whose cell graph is a path of length k. The next proposition builds upon the ideas of Berendsohn et al. [3], who proved a similar result for the class Av(321) using the fact that this class contains a staircase-shaped grid path of arbitrary length.

▶ **Proposition 3.** *If a permutation class $\mathcal{C}$ has the LPP then* $\mathrm{tw}_{\mathcal{C}}(n) \in \Omega(\sqrt{n})$.

Proof. First, we show that $\mathcal{C}$ contains for every k a grid subclass whose cell graph is a proper-turning path of length k, i.e. a path in which no three consecutive vertices are in the same row or column of the gridding. For the contrary, assume that there is ℓ such that $\mathcal{C}$ does not contain such path of length ℓ. The LPP then implies that $\mathcal{C}$ contains for every t a class $\mathrm{Grid}(\mathcal{M})$ where $\mathcal{M}$ is either a $1 \times t$ or $t \times 1$ matrix without empty entries. However, any such matrix of dimensions $1 \times n$ or $n \times 1$ contains all permutations of length n and thus, $\mathcal{C}$ must actually be the class of all permutations that contains all possible proper turning paths.

So we can suppose that there is a monotone gridding matrix $\mathcal{M}$ such that $\mathcal{M}$ is a proper-turning path $v_1, \ldots, v_{2m-1}$ of length $2m-1$ and $\mathrm{Grid}(\mathcal{M})$ is contained in $\mathcal{C}$. We explicitly construct a permutation $\pi \in \mathrm{Grid}(\mathcal{M})$ such that G_π contains an $m \times m$ grid graph as a subgraph. The claim then follows since the tree-width of $m \times m$ grid graph is exactly m.

For $i \in [m]$ and $j \in [i]$, let

$$p_{i,j} = (m + 2j - i - 1, m + 2j - i - 1), \quad p_{2m-i,j} = p_{i,j}.$$

We define a family of $2m$-tiles $\mathcal{P}$ by setting $\mathcal{P}_{v_i}$ to be the set of points $p_{i,j}$ for all possible choices of j.

Let $\mathcal{F}$ be a consistent orientation of $\mathcal{M}$ guaranteed by Lemma 1 and let π be the $\mathcal{F}$-assembly of $\mathcal{P}$. The sets P_{v_i} were defined in such a way that for every i the points in $P_{v_{2i}}$ have both coordinates odd whereas the points in $P_{v_{2i+1}}$ have both coordinates even. Since $\mathcal{M}$ is a proper turning path, there are always at most two non-empty tiles sharing the same row or column in π and in such case they correspond to neighboring vertices of the path. Moreover, if they share a common row, then the y-coordinates of their points are interleaved, and if they share a common column, the same holds for the x-coordinates.

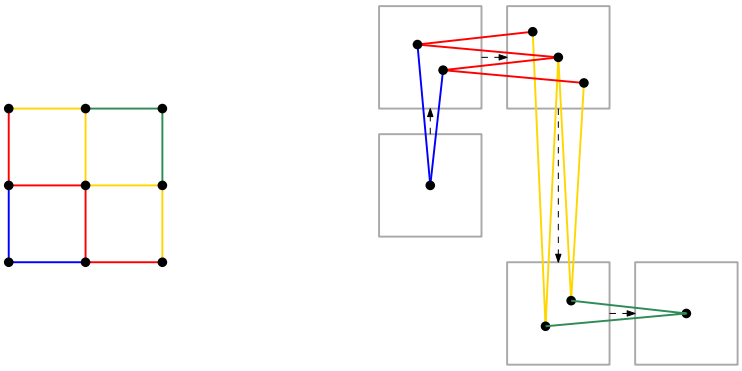


Figure 3 Illustration of the proof of Proposition 3. Embedding a 3×3 grid graph (left) into a permutation from a monotone grid class whose cell graph is a path of length 5 (right).

It remains to show that G_π contains an $m \times m$ grid graph as a subgraph. Let $s_{i,j}$ be the image of $p_{i,j}$ under the $\mathcal{F}$-assembly. We claim that we can map consecutive diagonals of the grid to the tiles P_{v_i}. See Figure 3. More precisely, for $x, y \in [m]$ set

$$g_{x,y} = \begin{cases} s_{x+y-1,x} & \text{if } x+y \leq m+1, \\ s_{x+y-1,m-y+1} & \text{otherwise.} \end{cases}$$

We start by showing that for any $i \in [m-1]$, there is an edge between $s_{i,j}$ and $s_{i+1,j}$, and also between $s_{i,j}$ and $s_{i+1,j+1}$. This follows since the points of P_{v_i} and $P_{v_{i+1}}$ have their x- or y-coordinates interleaved and there is no other tile occupying their shared row or column. Due to symmetry, it holds that for $i > m$, there is an edge between $s_{i,j}$ and $s_{i-1,j}$ and also between $s_{i,j}$ and $s_{i-1,j+1}$.

If we take $x, y \in [m]$ such that $x + y \leq m$ (i.e. $g_{x,y}$ lies below the anti-diagonal of the grid), the fact proved in the previous paragraph directly translates to the existence of edges between $g_{x,y}$ and $g_{x+1,y}$ and between $g_{x,y}$ and $g_{x,y+1}$. On the other hand for $x, y \in [m]$ such that $x + y \geq m + 2$, the points $g_{x,y}$, $g_{x-1,y}$ and $g_{x,y-1}$ translate to $s_{x+y-1,x}$, $s_{x+y-2,x-1}$ and $s_{x+y-2,x}$. Therefore, in this case there are edges between $g_{x,y}$ and $g_{x-1,y}$ and between $g_{x,y}$ and $g_{x,y-1}$. This concludes the proof as any edge in the $m \times m$ grid graph is of one of the two types whose existence we proved. ◀

It turns out that there is a large family of monotone grid classes for which $\mathrm{tw}_\mathcal{C} \in \Theta(\sqrt{n})$, namely every monotone grid class whose cell graph is not acyclic yet it does not contain two connected cycles. We include the complete proof in the full version, and here we only briefly describe its main ideas.

▶ **Theorem 4.** *If $\mathcal{M}$ is a connected monotone gridding matrix that contains a single cycle in its cell graph then* $\mathrm{tw}_{Grid(\mathcal{M})}(n) \in \Theta(\sqrt{n})$

Proof idea. First, we show that $\mathrm{tw}_{\mathrm{Grid}(\mathcal{M})}(n) \in \Omega(\sqrt{n})$. It has been previously proved by the authors in [12, Lemma 3.5] that a cycle in a grid class implies the LPP. The lower bound readily follows from Proposition 3.

For the upper bound, let π be a permutation of $\mathrm{Grid}(\mathcal{M})$ with a given $\mathcal{M}$-gridding. We observe that there is only $O(1)$ edges whose endpoints share neither a common row nor a common column. Therefore, we can focus on the graph G' obtained from G_π by removing these edges. We subsequently show that G' can be drawn on a surface of Euler genus 1 with $O(n)$ total crossings. Standard techniques [7] then imply that $\mathrm{tw}(G') \in O(\sqrt{n})$.

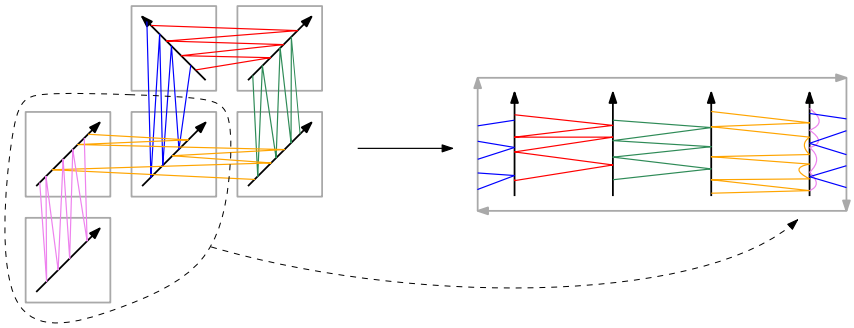


Figure 4 A schematic drawing of G_π for π from a unicyclic grid class on the projective plane. Instead of drawing the specific points of π, we place arrows to indicate the orientation of each cell. Different color is used for each set of edges that share a single row or column, and the exceptional edges are omitted.

Suppose that $c_1, c_2, \ldots, c_m$ are the entries of $\mathcal{M}$ that lie on its only cycle in this order. The cell graph $G_\mathcal{M}$ consists of the cycle and trees that are attached to it. If we remove all the edges that participate in the cycle, we end up with m trees $T_1, \ldots, T_m$ called *tendrils* such that the tree T_i contains the entry c_i.

We prove that the points of a single tendril can be drawn on a straight segment in a way such that the points from different cells are ordered consistently, and moreover, there are only $O(n)$ crossings between edges going inside a single tendril. We use this to draw each tendril on a parallel line, called *meridian*, and subsequently, we draw the edges connecting points in different tendrils as polylines that do not cross each other. We are forced to add one crosscap between some pair of meridians if $\mathcal{M}$ does not admit a consistent orientation. Finally, we check that we produced at most $O(n)$ crossing between the edges whose endpoints occupy a single tendril and the edges connecting two different tendrils. See Figure 4. ◀

For integer constants c and d, a *c-subdivided binary tree of depth d* is a graph obtained from a binary tree of depth d by replacing every edge by a path of length at most c. We say that a permutation class $\mathcal{C}$ has the *deep tree property* (DTP) if there is a constant c such that for every d, the class $\mathcal{C}$ contains a monotone grid subclass whose cell graph is a c-subdivided binary tree of depth d. Observe that DTP straightforwardly implies LPP. We say that a class $\mathcal{C}$ has *near-linear width* if $\mathrm{tw}_\mathcal{C}(n) \in \Omega(n/\log n)$.

▶ **Proposition 5.** *If a permutation class $\mathcal{C}$ has the DTP, then it has near-linear width.*

Proof. Inspired by the approach of Berendsohn [3], we want to show that for a graph G of large tree-width, we can find a permutation $\sigma \in \mathcal{C}$ such that G_σ contains G as a minor while the length of σ exceeds the size of G by at most a logarithmic factor.

To that end, fix an arbitrary graph G with vertex set $V_G = [n]$ and edges $\{e_1, \ldots, e_m\}$ where $e_i = \{a_i, b_i\}$. Let $\mathcal{M}$ be a monotone gridding matrix such that $\mathrm{Grid}(\mathcal{M}) \subseteq \mathcal{C}$ and the cell graph of $\mathcal{M}$ is a c-subdivided binary tree with exactly m leaves. Let r denote the root of this tree. It follows that the tree has maximal depth at most $c(\log m + 1)$. We turn $G_\mathcal{M}$ into an oriented graph by orienting all edges consistently away from r. For any vertex v of the tree, the *descendants of v*, denoted by $D(v)$, are all the out-neighbors of v.

We assign a set $A_w \subseteq V_G$ to each vertex w of the tree. First, we arbitrarily order the m leaves of $G_\mathcal{M}$ as $v_1, \ldots, v_m$. Then we inductively define

$$A_w = \begin{cases} \{a_i, b_i\} & \text{if } w = v_i \text{ for } i \in [m] \text{ where } e_i = \{a_i, b_i\}, \\ \bigcup_{v \in D(w)} A_v & \text{otherwise.} \end{cases} \tag{1}$$

We remark that $\sum_v |A_v| = O(m \log m)$ since each vertex $i \in V_G$ is present in exactly $\deg(i)$ leaves and in the paths of length $O(\log m)$ that connect those leaves to r. We proceed to define a family of m-tiles $\mathcal{P}$ by setting $P_v = \{(i,i) \mid i \in A_v\}$ for every vertex v of the tree, and keeping all the other tiles empty.

Let $\mathcal{F}$ be a consistent orientation obtained from the application of Lemma 1 on $\mathcal{M}$ and let π be the $\mathcal{F}$-assembly of $\mathcal{P}$. Since every tile is an increasing point set, it follows that π belongs to $\mathrm{Grid}(\mathcal{M})$.

In order to simplify the rest of the proof, we color S with n colors. We assign a color $i \in V_G$ to a point $p \in S$ with preimage (i,i) in $P_{x,y}$. We claim that S satisfies the following conditions:

(a) The subgraph of G_π induced by a single color is connected;

(b) For each edge $e_i = \{a_i, b_i\}$ of G there is an edge in G_π between a vertex of color a_i and a vertex of color b_i.

Fix a color $i \in V_G$. Let Q_i be the set of all vertices v of $G_\mathcal{M}$ such that $i \in A_v$. Clearly, Q_i induces a connected subtree of $G_\mathcal{M}$. Recall that every point of color i has always the coordinates (i,i) inside any tile. It follows that for points (i,i) in two neighboring tiles, the $\mathcal{F}$-assembly of $\mathcal{P}$ transforms them first to points that share one coordinate and then by rotating slightly clockwise makes them either horizontal or vertical neighbors. Therefore, the subgraph of G_π induced by color i is connected, which proves a.

Every leaf v_i must be the only non-empty vertex in its row or column. Let us assume the latter case as the other one is symmetric. Therefore, the two points contained in the image of P_{v_i} form an edge in G_π since no other point lies in the vertical strip between them. In particular, the leaf v_i satisfies the condition b for edge e_i.

The conditions a and b together imply that we can obtain a supergraph of G by contracting every monochromatic subgraph of G_π to a single vertex and thus, G is a minor of G_π. Observe that the total size of π is equal to $\sum_v |A_v|$ which we showed to be $O(m \log m)$. And since there exist graphs on n vertices with $O(n)$ edges and tree-width $\Omega(n)$, we deduce that $\mathrm{tw}_C(n) \geq \mathrm{tw}_{\mathrm{Grid}(\mathcal{M})}(n) \in \Omega(n/\log n)$. ◀

We continue by introducing a different property that implies the deep tree property and is at the same time easier to show for a specific class $\mathcal{C}$. A permutation class $\mathcal{C}$ has the *bicycle property* if it contains a monotone grid subclass whose cell graph is connected and contains at least two cycles. We include the full, rather technical, proof in the full version, providing here with only a brief sketch.

▶ **Proposition 6.** *If a permutation class $\mathcal{C}$ has the bicycle property, then it also has the DTP (and therefore near-linear width).*

Proof idea. The proof consists of two parts. First, we will show that there is always a grid subclass of $\mathcal{C}$ that contains in its cell graph two connected cycles of a certain special type. To that end, observe that the cell graph $G_\mathcal{M}$ can either be two cycles connected by a path or one cycle with a chord. For the latter case, we show that by replacing each entry in $\mathcal{M}$ with a suitable 3×3 matrix, we obtain a matrix $\mathcal{N}$ such that $\mathrm{Grid}(\mathcal{N})$ is a subclass of $\mathrm{Grid}(\mathcal{M})$ and moreover, the cell graph $G_\mathcal{N}$ contains two cycles joined with a path. See Figure 5.

In the second step, we find a way to wind a c-subdivided binary tree of arbitrary depth into the two cycles joined with a path and thus, showing that $\mathcal{C}$ has the DTP. ◀

For monotone grid classes, the results of this section imply a sharp dichotomy.

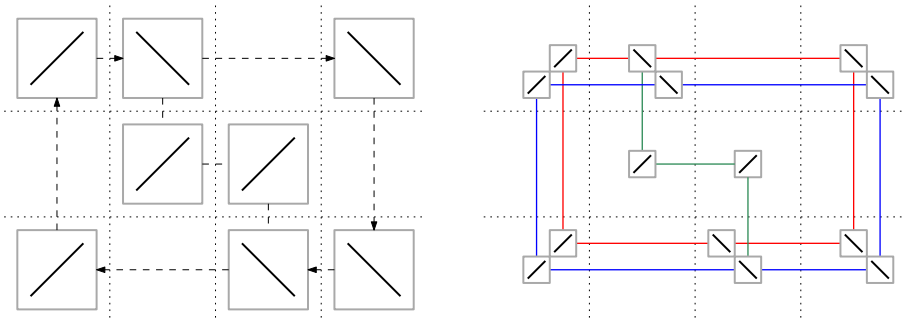


Figure 5 Left: a gridding matrix $\mathcal{M}$ whose cell graph is a cycle with a chord. Right: a gridding matrix $\mathcal{N}$ such that $\mathrm{Grid}(\mathcal{N})$ is contained in $\mathrm{Grid}(\mathcal{M})$ and the cell graph $G_{\mathcal{N}}$ consists of two cycles joined by a path.

▶ **Corollary 7.** *For a monotone grid class $\mathrm{Grid}(\mathcal{M})$ exactly one of the following holds.*

- *$G_{\mathcal{M}}$ is acyclic and $\mathrm{tw}_{\mathcal{C}}(k) \in \Theta(1)$.*
- *$G_{\mathcal{M}}$ contains at most one cycle in each component, $\mathcal{C}$ has LPP and $\mathrm{tw}_{\mathcal{C}}(k) \in \Theta(\sqrt{k})$.*
- *$G_{\mathcal{M}}$ has a component with at least two cycles, $\mathcal{C}$ has DTP and $\mathrm{tw}_{\mathcal{C}}(k) \in \Omega(k/\log k)$.*

3.2 The case of principal classes

In this section, we investigate the long path and deep tree properties of principal classes, i.e., the classes of the form $\mathrm{Av}(\pi)$. Combined with the results of Subsection 3.1, it allows us to infer lower bounds for the tree-width growth function of $\mathrm{Av}(\sigma)$. Whereas together with the results of Section 4, we obtain conditional lower bounds for counting patterns from $\mathrm{Av}(\sigma)$. Let us note that previously Berendsohn [2] has shown that for any π of length at least 4 that is not symmetric to one of $\{3412, 3142, 4213, 4123, 42153, 41352, 42513\}$, the class $\mathrm{Av}(\pi)$ has near-linear width. We reproduce and improve this result in a concise way with the tools that we have built up.

The *k-step increasing $(\mathcal{C}, \mathcal{D})$-staircase*, denoted by $\mathrm{St}_k(\mathcal{C}, \mathcal{D})$ is a grid class $\mathrm{Grid}(\mathcal{M})$ of a $k \times (k+1)$ gridding matrix $\mathcal{M}$ such that the only non-empty entries in $\mathcal{M}$ are $\mathcal{M}_{i,i} = \mathcal{C}$ and $\mathcal{M}_{i,i+1} = \mathcal{D}$ for every $i \in [k]$. In other words, the entries on the main diagonal are equal to $\mathcal{C}$ and the entries of the adjacent lower diagonal are equal to $\mathcal{D}$. The *increasing $(\mathcal{C}, \mathcal{D})$-staircase*, denoted by $\mathrm{St}(\mathcal{C}, \mathcal{D})$, is the union of $\mathrm{St}_k(\mathcal{C}, \mathcal{D})$ over all $k \in \mathbb{N}$.

The authors [13] recently showed that $\mathrm{Av}(\sigma)$ contains a certain staircase class for three patterns of length 3 and certain 2×2 grid classes for four patterns of length 4. Moreover, at least one of these patterns or their symmetries is contained in every permutation of length at least 4 that is not symmetric to one of 3412, 3142, 4213, 4123 or 41352.

▶ **Proposition 8** (Jelínek et al.[13]). *We have $\mathrm{St}(\boxslash, \mathrm{Av}(321)) \subseteq \mathrm{Av}(4321)$, $\mathrm{St}(\boxslash, \mathrm{Av}(231)) \subseteq \mathrm{Av}(4231)$ and $\mathrm{St}(\boxslash, \mathrm{Av}(312)) \subseteq \mathrm{Av}(4312)$.*

▶ **Proposition 9** (Jelínek et al.[13]). *The class $\mathrm{Av}(\sigma)$ contains the class $\mathrm{Grid}(\mathcal{M})$ for the gridding matrix $\mathcal{M} = \begin{pmatrix} \diagdown & \boxslash \\ \mathrm{Av}(\pi) & \diagdown \end{pmatrix}$ whenever*

- *$\pi = 132$ and $\sigma = 14523$, or*
- *$\pi = 231$ and $\sigma = 24513$, or*
- *$\pi = 321$ and $\sigma \in \{32154, 42513\}$.*

σ	LPP, DTP of $\mathrm{Av}(\sigma)$	Comment
1, 21, 312	neither LPP nor DTP	$\mathrm{tw}_{\mathrm{Av}(\sigma)} \in \Theta(1)$ by Ahal and Rabinovich [1] which would contradict Proposition 3.
321, 3412, 3142, 4213, 4123, 41352	LPP but not DTP	LPP of 321 and 3412 follows due to Jelínek and Kynčl [11], the rest contains either 123 or 321. The absence of DTP is proved in the full version.
All other	both LPP and DTP	DTP by Proposition 10, LPP follows.

Figure 6 The long path and deep tree properties of principal classes, i.e. classes of form $\mathrm{Av}(\sigma)$. Only one pattern σ from each symmetry group is listed.

▶ **Proposition 10.** *If σ is a permutation of length at least 4 that is not in symmetric to any of $3412, 3142, 4213, 4123$ or 41352, then $\mathrm{Av}(\sigma)$ has the bicycle property and thus, $\mathrm{Av}(\sigma)$ has near-linear width.*

Proof. We start by proving that every class defined by forbidding a pattern of length 3 must contain a special type of monotone grid subclass. For arbitrary π of length 3, the class $\mathrm{Av}(\pi)$ contains a grid class $\mathrm{Grid}(\mathcal{M})$ such that $\mathcal{M}$ is a 2×2 monotone gridding matrix with three non-empty entries. Since there are only two different symmetry types of permutations of length 3, it is enough to check that

$$\mathrm{Grid}\begin{pmatrix} \nearrow & \nearrow \\ \nearrow & \cdot \end{pmatrix} \subseteq \mathrm{Av}(321) \quad \text{and} \quad \mathrm{Grid}\begin{pmatrix} \searrow & \nearrow \\ \cdot & \searrow \end{pmatrix} \subseteq \mathrm{Av}(132).$$

First, we prove the claim for the patterns that appear in Proposition 8. Let $\sigma \in \{4321, 4231, 4312\}$ and take a 3-step increasing staircase $\mathrm{St}_3(\nearrow, \mathrm{Av}(\pi))$ for π of length 3 that is contained in $\mathrm{Av}(\sigma)$. Let $\mathcal{M}'$ be a 6×8 monotone gridding matrix obtained from $\mathrm{St}_3(\nearrow, \mathrm{Av}(\pi))$ by replacing every $\nearrow$-entry by the identity matrix $\begin{pmatrix} \cdot & \nearrow \\ \nearrow & \cdot \end{pmatrix}$ and every $\mathrm{Av}(\pi)$-entry with its 2×2 monotone grid subclass which has three non-empty entries. Clearly, $\mathrm{Grid}(\mathcal{M}')$ is a subclass of $\mathrm{Av}(\sigma)$, and it is easy to check that for any π, the cell graph of $\mathcal{M}'$ is connected and contains two cycles.

We prove the claim for the patterns that appear in Proposition 9 in a similar fashion. Let $\sigma \in \{14523, 24513, 32154, 42513\}$ and take $\mathcal{M}$ to be the grid class $\mathrm{Grid}\begin{pmatrix} \searrow & \nearrow \\ \mathrm{Av}(\pi) & \searrow \end{pmatrix}$ for π of length 3 that is contained in $\mathrm{Av}(\sigma)$. Similar to before, let $\mathcal{M}'$ be the gridding matrix obtained from $\mathcal{M}$ by replacing the $\nearrow$-entry with the matrix $\begin{pmatrix} \cdot & \nearrow \\ \nearrow & \cdot \end{pmatrix}$, both $\searrow$-entries with the matrix $\begin{pmatrix} \searrow & \cdot \\ \cdot & \searrow \end{pmatrix}$, and $\mathrm{Av}(\pi)$ with its 2×2 monotone grid subclass which has three non-empty entries. Again, $\mathrm{Grid}(\mathcal{M}')$ is a subclass of $\mathrm{Av}(\sigma)$, and it is easy to check that for any π, the cell graph of $\mathcal{M}'$ is connected and contains two cycles. ◀

We can actually show that the DTP cannot get us any further, since for any $\sigma \in \{3412, 3142, 4213, 4123, 41352\}$, the class $\mathrm{Av}(\sigma)$ does not have the DTP. See the full version for the whole discussion. Hereby, we actually obtained a complete knowledge of LPP and DTP for principal classes. See Figure 6.

4 Hardness of #PPM

In this section, we provide conditional lower bounds for modified variants of $\mathcal{C}$-Pattern PPM given LPP or DTP. The results of this section are proved under a slightly stronger assumptions about the classes. Apart from the LPP or DTP property, we furthermore require

an algorithm that provides a witnessing long path or deep tree. Formally, a class $\mathcal{C}$ has the *computable LPP* if it has the LPP and there is an algorithm that, for a given k, outputs the description of a monotone grid subclass of $\mathcal{C}$ whose cell graph is a path of length k. Similarly, a class $\mathcal{C}$ has the *computable DTP* if it has the DTP and there is an algorithm that, for a given k, outputs the description of a monotone grid subclass of $\mathcal{C}$ whose cell graph is a c-subdivided binary tree of depth k. Observe that all the specific examples of classes we encountered (and especially the principal classes in Subsection 3.2) posses the computable version of their corresponding properties.

We will reduce from the well-known problem *partitioned subgraph isomorphism* (PSI) defined as follows. We receive on input two graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$ together with a coloring $\chi\colon V_H \to V_G$ of vertices of H, using the vertices of G as colors. We have to decide if there is a mapping $\phi\colon V_G \to V_H$ such that whenever $\{u, v\} \in E_G$ then also $\{\phi(u), \phi(v)\} \in E_H$ and moreover $\chi(\phi(v)) = v$ for every $v \in V_G$. Less formally, we aim to find G as a subgraph of H, but we prescribe in advance where each vertex can be mapped to. It is a well-known fact that PSI is hard to solve.

▶ **Theorem 11** (Marx [14], Bringmann et al. [5]). *Unless ETH fails, PSI cannot be solved in time $f(k) \cdot n^{o(k/\log k)}$ for any function f, where $n = |V_H|$ and $k = |E_G|$. This is true even when we require G to have exactly as many vertices as edges.*

If we additionally fix G to be the clique on k vertices we obtain the problem called Partitioned Clique. Formally, the input to Partitioned Clique consists of a graph $H = (V_H, E_H)$ together with a coloring $\chi\colon V_H \to [k]$ and we have to decide if there is a k-clique in H that hits all k available colors. It is easy to see that Partitioned Clique can be solved in time $f(k) \cdot n^{O(k)}$. However, there is also a matching conditional lower bound.

▶ **Theorem 12** (Cygan et al. [6]). *Unless ETH fails, Partitioned Clique cannot be solved in time $f(k) \cdot n^{o(k)}$ for any function f, where $n = |V_H|$.*

We shall also not reduce directly to the problems of interest. Rather, we first reduce to the *$\mathcal{C}$-Pattern Anchored PPM* ($\mathcal{C}$-Pattern APPM) problem, defined as follows. The input consists of permutations $\pi \in \mathcal{C}$ and arbitrary τ together with pairs of points A in π and B in τ that are called *anchors*. We are promised that arbitrary inflation of the points in A with either two increasing or two decreasing sequences creates π' that is still contained in $\mathcal{C}$. The goal is to decide whether there is an embedding of π into τ that maps A to B.

For $\mathcal{C}$ with the computable LPP, we are able to reduce Partitioned Clique to $\mathcal{C}$-Pattern APPM such that the size of the pattern π is linear in the number of vertices of the clique. And for $\mathcal{C}$ with the computable DTP, we provide a reduction from PSI to $\mathcal{C}$-Pattern APPM such that the size of π is almost linear in the size of the graph G. Due to the space constraints and technicality of the reductions, we include here only brief overviews and describe both of them, including the proofs of correctness in the full version.

▶ **Lemma 13.** *Let $\mathcal{C}$ be a class with the computable LPP. An instance (G, χ) of Partitioned Clique can be reduced to an instance (π, τ, A, B) of $\mathcal{C}$-Pattern APPM where $|\pi| \in O(k^2)$ and $|\tau| \in O(|V_H|^2)$ in time $f(k) \cdot |V_H|^{O(1)}$ for some function f. Moreover, $\mathrm{tw}(\pi) \in O(k)$.*

Proof idea. Using the computable LPP, we obtain a monotone gridding matrix $\mathcal{M}$ such that $\mathrm{Grid}(\mathcal{M})$ is a subclass of $\mathcal{C}$ and the cell graph of $\mathcal{M}$ is a proper-turning path with $4k - 2$ vertices $v_1, \ldots, v_{4k-2}$. We construct the pattern π via an $\mathcal{F}$-assembly from a family of tiles $\mathcal{P}$ and the text τ from a family of tiles $\mathcal{T}$ where the only non-empty tiles in both families correspond to the non-empty entries of $\mathcal{M}$ and moreover, each non-empty tile in $\mathcal{P}$ is an increasing sequence.

The first tiles P_{v_1} and T_{v_1} both contain only pair of elements whose images under the $\mathcal{F}$-assembly are taken as the anchors A and B. Their role is to guarantee that any embedding of π into τ that maps A to B must be grid-preserving, i.e. it maps the image of the tile P_{v_i} to the image of the tile T_{v_i} for every i. The second pair of tiles P_{v_2} and T_{v_2} then simulates a mapping $\phi\colon [k] \to V_H$ that respects the coloring χ. And finally for every $i \in [k]$, the tiles corresponding to vertices v_{4a-1}, v_{4a}, v_{4a+1} and v_{4a+2} verify that there is an edge in H between the vertex $\phi(i)$ and $\phi(j)$ for every $j > i$. ◀

▶ **Lemma 14.** *Let $\mathcal{C}$ be a class with the computable DTP. An instance (G, H, χ) of PSI can be reduced to an instance (π, τ, A, B) of $\mathcal{C}$-Pattern APPM where $|\pi| \in O(|E_G| \cdot \log |E_G|)$ and $|\tau| \in O(|E_H| + |V_H| \cdot |E_G|)$ in time $f(|E_G|) \cdot |V_H|^{O(1)}$ for some function f.*

Proof idea. In this reduction, we combine the ideas of the reduction for the LPP (Lemma 13) with the proof that DTP implies near-linear tree-width (Proposition 5).

Using the computable DTP, we obtain a monotone gridding matrix $\mathcal{M}$ such that Grid($\mathcal{M}$) is a subclass of $\mathcal{C}$ and the cell graph of $\mathcal{M}$ is a c-subdivided binary tree with $|E_G|$ leaves. Additionally, we require that the root r of the tree has a single child r', and that each parent of a leaf has no other children. We again construct the pattern π via an $\mathcal{F}$-assembly from a family of tiles $\mathcal{P}$ and the text τ from a family of tiles $\mathcal{T}$ where the only non-empty tiles in both families correspond to the non-empty entries of $\mathcal{M}$ with each non-empty tile in $\mathcal{P}$ being an increasing sequence.

We set the tiles P_r and T_r to contain each a pair of elements which become the anchors A and B under the $\mathcal{F}$-assembly and which guarantee that any embedding of π into τ that respects the anchors must be grid-preserving. Using the same idea as in the proof of Lemma 13, the pair of tiles $P_{r'}$ and $T_{r'}$ is used to simulate a mapping $\phi\colon V_G \to V_H$ that respects the coloring χ. But now instead of verifying sequentially the neighborhood of each vertex in G, we aim to verify the existence of each edge in a particular leaf.

Set $k = |E_G|$. Following along the proof of Proposition 5, we orient the edges of the cell graph $G_{\mathcal{M}}$ consistently away from r and for any vertex v, the descendants of v, denoted by $D(v)$, are all the out-neighbors of v. We arbitrarily order the edges $E_G = \{e_1, \ldots, e_k\}$ and also the k leaves of $G_{\mathcal{M}}$ as $v_1, \ldots, v_k$, and we define the sets A_w exactly as in (1). We additionally assume that $A_r = [k]$ which corresponds to G having no isolated vertices. We again have $\sum_v A_v \in O(k \log k)$.

Now we spread the information about the mapping ϕ from r' to each leaf while keeping in each vertex only the information necessary to decide the existence of edges assigned to leaves in its subtree. In other words for a vertex v, we force the mapping of P_v into T_v to encode the mapping ϕ restricted to A_v. This in particular allows us to bound the size of π by $O(k \log k)$. Finally, we use the leaf v_i and its parent to test the existence of an edge $\{\phi(a_i), \phi(b_i)\} \in E_H$ where $e_i = \{a_i, b_i\}$ using the same construction as in Lemma 13. ◀

Observe that both reductions produce π and τ as gridded permutations belonging to some monotone grid class Grid($\mathcal{M}$) via an $\mathcal{F}$-assembly from families of tiles. Importantly, they share the property that any embedding of π into τ that maps A to B must be grid-preserving, i.e., it maps the (i, j)-cell of the gridding of π to th (i, j)-cell of the gridding of τ for every i and j. Moreover, both A and B are pairs of consecutive points in the left-to-right order.

4.1 Consequences

▶ **Theorem 15.** *If $\mathcal{C}$ has the computable long-path property then $\mathcal{C}$-Pattern PPM cannot be solved in time $f(t) \cdot n^{o(t)}$ where $t = \mathrm{tw}(\pi)$ for any function f, unless ETH fails.*

Proof. Let (π, τ, A, B) be the instance of C-Pattern APPM produced by Lemma 13 and let m be the length of τ. We define π' as the permutation obtained from π by inflating both of the anchors in A with either two increasing or decreasing sequences of length m such that π' is still contained in $\mathcal{C}$. Recall that one of these inflations is always possible. And similarly, we let τ' be the permutation obtained from τ by inflating both of the anchors in B with the same type of monotone sequences of length m as in π'.

We claim that π' is contained in τ' if and only if (π, τ, A, B) is a positive instance of C-Pattern APPM. It is clear that if there is an embedding of π into τ that maps A to B, then there is an embedding of π' into τ'.

For the other direction, assume there is an embedding ϕ of π' into τ'. The inflated anchors in π' contain exactly $2m$ points while τ' contains only $m-2$ points outside of its inflated anchors. Therefore, at least $m+2$ points of the inflated anchors in π' are mapped by ϕ to the inflated anchors in τ' and in particular, there must be at least one point in each of the anchors in π' mapped to the corresponding anchor in τ'. Since the anchors A and B are pairs of consecutive points, observe that we can, in fact, map the whole inflated anchors in π' to the inflated anchors in τ'. It follows that we obtain a desired anchored embedding of π into τ by deflating the anchors back to a single point.

Finally, we show that $\mathrm{tw}(\pi') \leq \mathrm{tw}(\pi) + 2$. The desired bound follows as otherwise, we could use a faster algorithm for $\mathcal{C}$-Pattern PPM to decide the instance (π, τ, A, B) of C-Pattern APPM and consequently refute ETH by the "moreover" part of Lemma 13. We claim that in general, if σ' is obtained from σ by inflating one point with a monotone sequence then $\mathrm{tw}(\sigma') \leq \mathrm{tw}(\sigma) + 1$. To see that, notice that when we inflate a point of σ with a monotone sequence of length 2, we get σ' such that $\mathrm{tw}(\sigma') \leq \mathrm{tw}(\sigma) + 1$. However, if we inflate the same point by a longer monotone sequence and obtain a permutation σ'' then $G_{\sigma''}$ can be obtained by edge subdivisions from $G_{\sigma'}$, and it is well-known that subdividing en edge does not increase tree-width. ◀

In order to show the hardness of $\mathcal{C}$-Pattern #PPM, we first reduce to an intermediate problem called *$\mathcal{C}$-Pattern Surjective Colored PPM* ($\mathcal{C}$-Pattern SCPPM) whose input consists of a pattern $\pi \in \mathcal{C}$, a text τ and a coloring $\chi \colon \tau \to [t]$. We need to decide whether there is an embedding of π into τ that hits all t possible colors. This intermediate reduction allows us to infer conditional lower bounds for $\mathcal{C}$-Pattern #PPM via the following lemma.

▶ **Lemma 16** (Berendsohn [2]). *Let there be an algorithm that solves $\mathcal{C}$-Pattern #PPM in time $f(k) \cdot n^{O(g(k))}$ for some functions f and g. Then $\mathcal{C}$-Pattern SCPPM can be solved in time $h(k) \cdot n^{O(g(k))}$ for some function h.*

▶ **Lemma 17.** *An instance (π, τ, A, B) of $\mathcal{C}$-Pattern APPM produced by Lemma 13 or 14 can be reduced to an instance (π', τ', χ) of $\mathcal{C}$-Pattern SCPPM where $|\pi'| \in O(|\pi|)$ and $|\tau'| \in O(|\tau|)$ in polynomial time.*

Proof. The general idea of the proof is the same as in Theorem 15 – we force matching of the anchors by inflating them with long monotone sequences. The $\mathcal{C}$-Pattern SCPPM problem, however, allows us to use sequences with length depending only on π. Let k be the length of π and let π' be the permutation obtained by inflating the anchors A with either two increasing or decreasing sequences of length k such that $\pi' \in \mathcal{C}$, and let τ' be the permutation obtained by the same inflation of the anchors B. We define $\chi \colon \tau \to [2k+1]$ by coloring every point added during the inflation with a unique color and using a single additional color for every other point. Clearly, $|\pi'| \in O(|\pi|)$ and $|\tau'| \in O(|\tau|)$.

We need to verify the correctness of our construction. If (π, τ, A, B) is a positive instance of $\mathcal{C}$-Pattern APPM then (π', τ', χ) is a positive instance of $\mathcal{C}$-Pattern SCPPM as we can simply map the inflated anchors of π' to the inflated anchors of τ'. For the other

direction, assume there is an embedding ϕ of π' into τ' that hits all the $2k+1$ colors. In other words, the image of π' under ϕ contains the whole inflated anchors of τ'. Since there are only $k-2$ points in π' outside of the anchors, at least $k+2$ points of the anchors in π' maps to the anchors in τ'. In particular, there must be at least one point in each of the two increasing inflated anchors in π' that maps to the corresponding anchor in τ'. By the same argument as in the proof of Theorem 15, we conclude that the inflated anchors map without loss of generality to the inflated anchors. ◀

▶ **Theorem 18.** *Unless ETH fails, $\mathcal{C}$-Pattern #PPM cannot be solved for any function f*

- *in time $f(k) \cdot n^{o(\sqrt{k})}$ if $\mathcal{C}$ has the computable LPP, and*
- *in time $f(k) \cdot n^{o(k/\log^2 k)}$ if $\mathcal{C}$ has the computable DTP.*

Proof. For $\mathcal{C}$ with the computable LPP, a faster algorithm would refute ETH via

$$\text{Partitioned Clique} \xrightarrow{\text{Lemma 13}} \mathcal{C}\text{-Pattern APPM} \xrightarrow{\text{Lemma 17}} \mathcal{C}\text{-Pattern SCPPM} \xrightarrow{\text{Lemma 16}} \mathcal{C}\text{-Pattern \#PPM}$$

Whereas for $\mathcal{C}$ with the computable DTP, a faster algorithm would refute ETH via

$$\text{PSI} \xrightarrow{\text{Lemma 14}} \mathcal{C}\text{-Pattern APPM} \xrightarrow{\text{Lemma 17}} \mathcal{C}\text{-Pattern SCPPM} \xrightarrow{\text{Lemma 16}} \mathcal{C}\text{-Pattern \#PPM.}$$ ◀

References

1. Shlomo Ahal and Yuri Rabinovich. On complexity of the subpattern problem. *SIAM J. Discrete Math.*, 22(2):629–649, 2008. doi:10.1137/S0895480104444776.
2. Benjamin Aram Berendsohn. Complexity of permutation pattern matching. Master's thesis, Freie Universität Berlin, Berlin, 2019.
3. Benjamin Aram Berendsohn, László Kozma, and Dániel Marx. Finding and counting permutations via CSPs. In Bart M. P. Jansen and Jan Arne Telle, editors, *14th International Symposium on Parameterized and Exact Computation, IPEC 2019, September 11-13, 2019, Munich, Germany*, volume 148 of *LIPIcs*, pages 1:1–1:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.IPEC.2019.1.
4. Prosenjit Bose, Jonathan F. Buss, and Anna Lubiw. Pattern matching for permutations. *Inform. Process. Lett.*, 65(5):277–283, 1998. doi:10.1016/S0020-0190(97)00209-3.
5. Karl Bringmann, László Kozma, Shay Moran, and N. S. Narayanaswamy. Hitting set for hypergraphs of low VC-dimension. In *24th Annual European Symposium on Algorithms*, volume 57 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 23, 18. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2016.
6. Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*. Springer, Cham, 2015. doi:10.1007/978-3-319-21275-3.
7. Vida Dujmović, David Eppstein, and David R. Wood. Structure of graphs with locally restricted crossings. *SIAM Journal on Discrete Mathematics*, 31(2):805–824, 2017. doi:10.1137/16M1062879.
8. Jacob Fox. Stanley–Wilf limits are typically exponential. arXiv:1310.8378v1, 2013.
9. Sylvain Guillemot and Dániel Marx. Finding small patterns in permutations in linear time. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 82–101. ACM, New York, 2014. doi:10.1137/1.9781611973402.7.
10. Sylvain Guillemot and Stéphane Vialette. Pattern matching for 321-avoiding permutations. In *Algorithms and computation*, volume 5878 of *Lecture Notes in Comput. Sci.*, pages 1064–1073. Springer, Berlin, 2009. doi:10.1007/978-3-642-10631-6_107.
11. Vít Jelínek and Jan Kynčl. Hardness of permutation pattern matching. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 378–396. SIAM, Philadelphia, PA, 2017. doi:10.1137/1.9781611974782.24.

12 Vít Jelínek, Michal Opler, and Jakub Pekárek. A complexity dichotomy for permutation pattern matching on grid classes. In Javier Esparza and Daniel Král', editors, *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic*, volume 170 of *LIPIcs*, pages 52:1–52:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.MFCS.2020.52.

13 Vít Jelínek, Michal Opler, and Jakub Pekárek. Griddings of Permutations and Hardness of Pattern Matching. In Filippo Bonchi and Simon J. Puglisi, editors, *46th International Symposium on Mathematical Foundations of Computer Science (MFCS 2021)*, volume 202 of *LIPIcs*, pages 65:1–65:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.MFCS.2021.65.

14 Dániel Marx. Can you beat treewidth? *Theory Comput.*, 6:85–112, 2010. doi:10.4086/toc.2010.v006a005.

15 Vincent Vatter and Steve Waton. On partial well-order for monotone grid classes of permutations. *Order*, 28(2):193–199, 2011. doi:10.1007/s11083-010-9165-1.

A Polynomial Kernel for Bipartite Permutation Vertex Deletion

Lawqueen Kanesh
National University of Singapore, Singapore

Jayakrishnan Madathil
Chennai Mathematical Institute, Chennai, India

Abhishek Sahu
The Institute of Mathematical Sciences, HBNI, Chennai, India

Saket Saurabh
The Institute of Mathematical Sciences, HBNI, Chennai, India
University of Bergen, Norway

Shaily Verma
The Institute of Mathematical Sciences, HBNI, Chennai, India

Abstract

In a permutation graph, vertices represent the elements of a permutation, and edges represent pairs of elements that are reversed by the permutation. In the Permutation Vertex Deletion problem, given an undirected graph G and an integer k, the objective is to test whether there exists a vertex subset $S \subseteq V(G)$ such that $|S| \leq k$ and $G - S$ is a permutation graph. The parameterized complexity of Permutation Vertex Deletion is a well-known open problem. Bożyk et al. [IPEC 2020] initiated a study towards this problem by requiring that $G - S$ be a bipartite permutation graph (a permutation graph that is bipartite). They called this the Bipartite Permutation Vertex Deletion (BPVD) problem. They showed that the problem admits a factor 9-approximation algorithm as well as a fixed parameter tractable (FPT) algorithm running in time $\mathcal{O}(9^k|V(G)|^9)$. And they posed the question *whether* BPVD *admits a polynomial kernel.*

We resolve this question in the affirmative by designing a polynomial kernel for BPVD. In particular, we obtain the following: Given an instance (G, k) of BPVD, in polynomial time we obtain an equivalent instance (G', k') of BPVD such that $k' \leq k$, and $|V(G')| + |E(G')| \leq k^{\mathcal{O}(1)}$.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases kernelization, bipartite permutation graph, bicliques

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.23

Funding *Lawqueen Kanesh*: Supported in part by NRF Fellowship for AI grant [R-252-000-B14-281] and by Defense Service Organization, Singapore.
Jayakrishnan Madathil: Supported by the Chennai Mathematical Institute and the Infosys Foundation.
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 Swarnajayanti Fellowship (no. DST/SJF/MSA01/2017-18).

1 Introduction

In a *graph modification problem*, the input consists of an n-vertex graph G and an integer k. The objective is to determine whether k *modification operations* – such as vertex deletions, or edge deletions, insertions or contractions – are sufficient to obtain a graph with prescribed structural properties such as being planar, bipartite, chordal, interval, acyclic or edgeless. Graph modification problems include some of the most basic problems in graph theory and

graph algorithms. Unfortunately, most of these problems are NP-complete [25, 33]. Therefore, they have been studied intensively within various algorithmic paradigms for coping with NP-completeness [14, 17, 27], including approximation algorithms, parameterized complexity, and algorithms for restricted input classes.

Graph modification problems have played a central role in the development of parameterized complexity. Here, the number of allowed modifications, k, is considered a *parameter*. With respect to k, we seek a *fixed parameter tractable* (FPT) algorithm, namely, an algorithm whose running time has the form $f(k)n^{\mathcal{O}(1)}$ for some computable function f. One way to obtain such an algorithm is to exhibit a *kernelization algorithm*, (or *kernel*, for short). A kernel for a graph problem Π is an algorithm that given an instance (G, k) of Π, runs in polynomial time and outputs an equivalent instance (G', k') of Π such that $|V(G')|$ and k' are upper bounded by $f(k)$ for some computable function f. The function f is called the *size* of the kernel, and if f is a polynomial function, then we say that the kernel is a *polynomial kernel*. A kernel for a problem immediately implies that it admits an FPT algorithm, but kernels are also interesting in their own right. In particular, kernels allow us to model the performance of polynomial time pre-processing algorithms. The field of kernelization has received considerable attention, especially after the introduction of the methods for proving kernelization lower bounds [3, 7, 8, 11, 16, 20, 21]. We refer to the surveys [15, 19, 24, 26], as well as the books [6, 10, 12, 30], for a detailed treatment of the area of kernelization. In this paper, we study the kernelization complexity of the following problem.

Bipartite Permutation Vertex Deletion (BPVD) **Parameter:** k
Input: A graph G and an integer k.
Question: Does there exist a subset $S \subseteq V(G)$ of size at most k such that $G - S$ is a bipartite permutation graph?

A graph G is a *permutation graph*, if the vertices represent the elements of a permutation, and edges represent pairs of elements that are reversed by the permutation. Alternatively, a permutation graph can be defined as an intersection graph of line segments whose endpoints lie on two parallel lines $\mathcal{L}_1$ and $\mathcal{L}_2$, with one endpoint of each line segment lying on $\mathcal{L}_1$ and the other endpoint on $\mathcal{L}_2$. Due to their intriguing combinatorial properties and modelling power, the class of permutation graphs is one of the well-studied graph classes [5, 18]. As a subclass of perfect graphs, many problems that are NP-complete on general graphs can be solved efficiently on permutation graphs, such as Clique, Independent Set, Chromatic Number, Treewidth and Pathwidth. Further, there is a linear time algorithm to test whether a given graph is a permutation graph, and if so construct a permutation representing it [29]. Whether Permutation Vertex Deletion admits an FPT algorithm has been a longstanding open problem in the area. In order to make progress on this open problem, recently, Bożyk et al. [4] studied the problem of deleting vertices to a subclass of permutation graphs. The subclasses of permutation graphs include the classes of bipartite permutation graphs (characterized by Spinrad, Brandstädt & Stewart 1987 [31]) and cographs. While the fixed-parameter tractability of vertex deletion to cographs follows easily because of the finite forbidden characterization (as induced subgraphs) of cographs, no such result was known for vertex deletion to bipartite permutation graphs. Bożyk et al. [4] studied BPVD, and showed that the problem admits a factor 9-approximation algorithm as well as a FPT algorithm running in time $\mathcal{O}(9^k n^9)$. A natural follow-up question to this work, explicitly asked in [4], is whether BPVD admits a polynomial kernel. In this paper, we resolve this question in the affirmative.

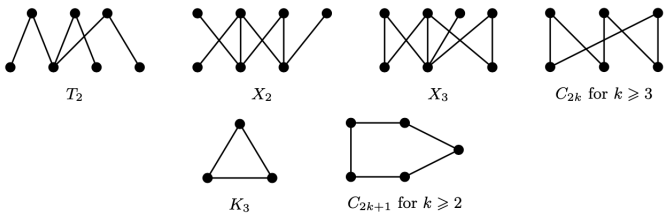


Figure 1 The set of obstructions for a bipartite permutation graph (Figure from [4]).

▶ **Theorem 1.** Bipartite Permutation Vertex Deletion *admits a polynomial kernel.*

1.1 Methods

Our kernelization heavily uses the characterization of bipartite permutation graphs in terms of their *forbidden induced subgraphs*, also called *obstructions*. Specifically, a graph H is an obstruction to the class of bipartite permutation graphs if H is not a bipartite permutation graph and $H - \{v\}$ is a bipartite permutation graph for every vertex $v \in V(H)$. A graph G is a bipartite permutation graph if and only if it does not contain any obstruction as an induced subgraph. The set of obstructions to bipartite permutation graphs have been completely characterized by Spinrad et al. [31]. It consists of T_2, X_2, X_3, K_3, as well two infinite families of graphs: even cycles of length at least 6, and odd cycles of length at least 5 (see Figure 1). We call any obstruction of size less than 45 a *small obstruction*, and call all other obstructions large obstructions. Note that every large obstruction is a hole (induced cycle) of length at least 45.

The first ingredient of our kernelization algorithm is the factor 9 polynomial time approximation algorithm for BPVD by Bożyk et al. [4]. We use this algorithm to obtain an approximate solution of size at most $9k$, or conclude that no solution of size at most k exists. We grow this approximate solution to a solution T of size $\mathcal{O}(k^{45})$, such that every set $Y \subseteq V(G)$ of size at most k is a minimal hitting set for small obstructions in G if and only if Y is a minimal hitting set for small obstructions in $G[T]$. Once we have T (also called a modulator), we know that $G - T$ is a bipartite permutation graph. Let S be a minimal (or minimum) solution of size at most k. Then, the only purpose of vertices in $S \cap (V(G) \setminus T)$ is to hit large obstructions. Next, we analyze the graph $G - T$, and reduce its size by applying various reduction rules.

For the kernelization algorithm, we look at $G - T$, and focus on *one* connected component of $G - T$. Since $G - T$ is a bipartite permutation graph, it has a "complete bipartite decomposition" [32]. For our kernelization purpose, we heavily use this known decomposition. A *biclique* or a *complete bipartite graph* is a bipartite graph where every vertex of the first part is adjacent to every vertex of the second part. We give a semi-formal definition of a complete bipartite decomposition [32]. Let $H = G - T$ and π be an ordering of $V(H)$. A sequence of vertex subsets $(Q_1, R_1, Q_2, R_2, \ldots, Q_s, R_s)$, where $Q_i, R_i \subseteq V(H)$ for every $i \in [s]$, is said to be a *complete bipartite decomposition* of H if the following holds. The vertex subsets partition $V(H)$, $H[Q_i]$ is a biclique for every $i \in [s]$, R_i is an independent set for every $i \in [s]$, and $Q_1 <_\pi R_1 <_\pi Q_2 <_\pi R_2 <_\pi \cdots <_\pi Q_s <_\pi R_s$. That is, if $X <_\pi Y$, then every vertex in X appears before every vertex in Y in π. Further, for $i, j \in [s]$, if $E(Q_i, Q_j) \neq \emptyset$, then $|i - j| \leq 1$, for $i, j \in [s]$, if $E(Q_i, R_j) \neq \emptyset$, then $i = j$, and for $i, j \in [s]$ with $i \neq j$, we have $E(R_i, R_j) = \emptyset$. Here, $E(X, Y)$ denotes the set of edges with one endpoint in X and the other in Y. Informally, $(Q_1, R_1, Q_2, R_2, \ldots, Q_s, R_s)$, is a partition of $V(H)$, where each part

is either a biclique or an independent set, and each set has edges only in the neighboring parts. The complete bipartite decomposition is similar to the clique partition used by Ke et al. [23] for designing a polynomial kernel for vertex deletion to proper interval graphs.

In the first phase, we bound the maximum biclique size in $G - T$, i.e., the size of Q_i for $i \in [s]$. Our biclique-reduction procedure builds upon the clique-reduction procedure of Marx [28], which was used in the kernelizations for Chordal Vertex Deletion [1, 22] and Interval Vertex Deletion [2]. The procedure of Marx [28] as well as our procedure are based on an "irrelevant vertex rule". In particular, we find a vertex that is not necessary for a solution of size at most k, and delete it. And after this procedure we reduce the size of each biclique in $G - T$ by $k^{\mathcal{O}(1)}$. Next, using a simple marking procedure we bound the size of R_i for $i \in [s]$ as well.

In the second phase we bound the size of the connected component of $G - T$ we started with. Towards this, we first bound the number of bicliques in $Q_1, Q_2, \ldots, Q_t$ that contain a neighbor of a vertex in T (say *good bicliques*). We use small obstructions, and in particular T_2 (the subdivided claw), and K_3 (the triangle) to bound the number of good bicliques by $k^{\mathcal{O}(1)}$. This automatically divides the biclique partition into *chunks*. Mark all the good bicliques. A maximal set of unmarked bicliques between two marked bicliques form a chunk. It is clear that the number of chunks is upper bounded by $k^{\mathcal{O}(1)}$. Finally, we use structural analysis to bound the size of each chunk, which includes the design of a reduction rule that computes a minimum cut between the two good bicliques that border the chunk. In particular, we show that each chunk can be replaced by a graph of size $k^{\mathcal{O}(1)}$. We remark that the procedure also needs to handle the presence of independent sets $R_1, R_2, \ldots, R_s$, which we have completely ignored in the discussion here.

Until now we have assumed that $G - T$ is connected. Finally, again using the obstructions T_2 and K_3, we show that the number of connected components in $G - T$ is upper bounded by $k^{\mathcal{O}(1)}$. Using this bound, together with the facts that $|T| \leq k^{\mathcal{O}(1)}$, and that each connected component is of size $k^{\mathcal{O}(1)}$, we are able to deduce our polynomial kernel for BPVD.

2 Preliminaries

In this section, we define some notations and list some properties of bipartite permutation graphs.

Standard Notation. For a positive integer n, we denote the set $\{1, 2, \ldots, n\}$ by $[n]$. For a graph G, $V(G)$ and $E(G)$ denote the set of vertices and edges, respectively. Two vertices u, v are said to be *adjacent* if there is an edge (denoted as uv) between u and v. Given vertex subsets $X, Y \subseteq V(G)$, such that $X \cap Y = \emptyset$, $E(X, Y)$ denotes the set of edges with one endpoint in X and the other in Y. The neighbourhood of a vertex v, denoted by $N_G(v)$, is the set of vertices adjacent to v. The subscript in the notation for neighbourhood is omitted, if the graph under consideration is clear. For a set $M \subseteq V(G)$ and a vertex $u \in V(G)$, by $M(u)$ we denote $N(u) \cap M$. For a set $S \subseteq V(G)$, $G - S$ denotes the graph obtained by deleting S from G and $G[S]$ denotes the subgraph of G induced on S. A *path* $P = v_1, \ldots, v_\ell$ is a sequence of distinct vertices where every consecutive pair of vertices is adjacent. We say that P *starts* at v_1 and *ends* at v_ℓ. The vertices (or vertex set) of P, denoted by $V(P)$, is the set $\{v_1, \ldots, v_\ell\}$. The *endpoints* of P is the set $\{v_1, v_\ell\}$ and the *internal vertices* of P is the set $V(P) \setminus \{v_1, v_\ell\}$. The *length* of P is defined as $|V(P)|$. A *cycle* is a sequence $v_1, \ldots, v_\ell$ of vertices such that $v_1, \ldots, v_\ell$ is a path and $v_\ell v_1$ is an edge. A set $Q \subseteq V(G)$ of pairwise adjacent vertices in G is called a *clique*. For graph theoretic terms and definitions not stated explicitly here, we refer to [9].

2.1 Bipartite permutation graph

The characterization of bipartite permutation graphs presented below was proposed by Spinrad et al. [31]. Let G be a connected bipartite graph with vertex bipartition (A, B). A linear order $(B, <_B)$ satisfies the *adjacency property* if for each vertex $u \in A$ the set $N(u)$ consists of vertices that are consecutive in $(B, <_B)$. A linear order $(B, <_B)$ satisfies the *enclosure property* if for every pair of vertices $u, u' \in A$ such that $N(u)$ is a subset of $N(u')$, vertices in $N(u') \setminus N(u)$ occur consecutively in $(B, <_B)$. A *strong ordering* of the vertices of $A \cup B$ consists of linear orders $(A, <_A)$ and $(B, <_B)$ such that for every $(u, w'), (u', w)$ in $E(G)$, where u, u' are in A and w, w' are in B, $u <_A u'$ and $w <_B w'$ imply that $(u, w) \in E(G)$ and $(u', w') \in E(G)$. Note that whenever $(A, <_A)$ and $(B, <_B)$ form a strong ordering of $A \cup B$, then $(A, <_A)$ and $(B, <_B)$ satisfy the adjacency and enclosure properties.

▶ **Theorem 2** ([31]). *The following three statements are equivalent for a connected bipartite graph $G = (A, B, E)$:*

1. *(A, B, E) is a bipartite permutation graph.*
2. *There exists a strong ordering of $A \cup B$.*
3. *There exists a linear order $(B, <_B)$ of B satisfying adjacency and enclosure properties.*

Notation on ordering. Let G be a bipartite permutation graph with a vertex bipartition, say (A, B), of G. Fix a strong ordering, say π, of (A, B). Let π_A and π_B be the restriction of π on A and B, respectively, that is, π_A and π_B are linear orderings of the vertices of A and B. For $X \in \{A, B\}$ and a pair of vertices $x, y \in X$, we say $x <_{\pi_X} y$ if x appears before y in the ordering π_X. Similarly, for $X \in \{A, B\}$ and $Y, Y' \subseteq X$, we say $Y <_\pi Y'$ if $y <_{\pi_X} y'$ for every $y \in Y$ and $y' \in Y'$. More generally, for $Y, Y' \subseteq A \cup B$, we write $Y <_\pi Y'$ if $Y \cap A <_\pi Y' \cap A$ and $Y \cap B <_\pi Y' \cap B$. For $X \in \{A, B\}$, a set $Y \subseteq X$ and an integer q, where $1 \leq q \leq |Y|$, we write F_q^Y to denote the first q vertices of Y in the ordering π_X. Similarly, we write L_q^Y to denote the last q vertices of Y in the ordering π_X.

2.2 Complete Bipartite Decomposition

We start by defining the notion of complete bipartite decomposition.

▶ **Definition 3** (Complete Bipartite Decomposition,[32]). *Consider a bipartite permutation graph G with vertex bipartition (A, B) and a strong ordering π of (A, B). A sequence of vertex subsets $(Q_1, R_1, Q_2, R_2, \ldots, Q_s, R_s)$, where $Q_i, R_i \subseteq V(G)$ for every $i \in [s]$, is said to be a* complete bipartite decomposition *of G if the following properties hold:*

1. *$\{Q_1, R_1, Q_2, R_2, \ldots, Q_s, R_s\}$ is a partition of $V(G)$,*
2. *$G[Q_i]$ is a biclique for every $i \in [s]$,*
3. *R_i is an independent set for every $i \in [s]$,*
4. *$Q_1 <_\pi R_1 <_\pi Q_2 <_\pi R_2 <_\pi \cdots <_\pi Q_s <_\pi R_s$,*
5. *for $i, j \in [s]$, if $E(Q_i, Q_j) \neq \emptyset$, then $|i - j| \leq 1$,*
6. *for $i, j \in [s]$, if $E(Q_i, R_j) \neq \emptyset$, then $i = j$,*
7. *for $i, j \in [s]$, we have $E(R_i, R_j) = \emptyset$.*

The next lemma proves that every connected bipartite permutation graph has a complete bipartite decomposition and further, it can be computed in polynomial time.

▶ **Lemma 4** ([32]). *Every connected bipartite permutation graph has a complete bipartite decomposition. Moreover, there is a polynomial time algorithm that takes a connected bipartite permutation graph G with a fixed vertex bipartition (A, B) and a fixed strong ordering π of (A, B) as input, and returns a complete bipartite decomposition of G.*

3 Constructing a Nice Modulator

We classify the set of obstructions for bipartite permutation graphs as follows. Any obstruction of size less than 45 is known as a *small obstruction*, while other obstructions (holes) are said to be large. In this section we design a modulator, with some additional properties, of size $k^{\mathcal{O}(1)}$ to bipartite permutation graph. For this we will utilize the following known results.

▶ **Theorem 5** ([4]). *There exists a factor* 9*-approximation algorithm for* BPVD.

▶ **Lemma 6** ([13, Lemma 3.2]). *Let $\mathcal{F}$ be a family of sets of cardinality at most d over a universe $\mathcal{U}$ and k be a positive integer. Then there is an $\mathcal{O}(|\mathcal{F}|(k+|\mathcal{F}|))$ time algorithm that finds a non-empty set $\mathcal{F}' \subseteq \mathcal{F}$ such that*

1. *For every $Z \subseteq \mathcal{U}$ of size at most k, Z is a minimal hitting set of $\mathcal{F}$ if and only if Z is a minimal hitting set of $\mathcal{F}'$; and*
2. *$|\mathcal{F}'| \leq d!(k+1)^d$.*

We use Lemma 6 to identify a vertex subset of $V(G)$, which allows us to forget about small induced subgraphs of G, and to concentrate on long induced holes for the kernelization.

▶ **Lemma 7 (♣).** [1] *Let (G,k) be an instance to* BPVD. *In polynomial time, we construct a vertex subset $T'' \subseteq V(G)$ such that*

1. *Every set $Y \subseteq V(G)$ of size at most k is a minimal hitting set of small obstructions in G if and only if it is a minimal hitting set for small obstructions in $G[T'']$, and*
2. *$|T''| \leq (45+1)!(k+1)^{45}$,*

or conclude that (G,k) is a no-instance.

Using Theorem 5, in polynomial time we construct a 9-approximate solution T', and using Lemma 7 in polynomial time we construct a vertex set T''. If $|T'| > 9k$ or $|T''| > (45+1)!(k+1)^{45}$, we conclude (G,k) is a no-instance. Otherwise we have a modulator $T = T' \cup T''$ of size $\mathcal{O}(k^{45})$, such that $G - T$ is a bipartite permutation graph, and every set $Y \subseteq V(G)$ of size at most k is a minimal hitting set of small obstructions in G if and only if it is a minimal hitting set for small obstructions in $G[T]$. Let S be a minimal (or minimum) solution of size at most k. Then, the only purpose of vertices in $S \cap (V(G) \setminus T)$ is to hit long obstructions. We call the modulator constructed above as *nice modulator*. We summarize these discussions in the next lemma.

▶ **Lemma 8** (Nice Modulator). *Let (G,k) be an instance to* BPVD. *In polynomial time, we can either construct a nice modulator $T \subseteq V(G)$ of size $\mathcal{O}(k^{45})$, or conclude that (G,k) is a no-instance.*

Furthermore, in $G - T = (A \cup B, <)$, $<$ is a strong ordering of the bipartite permutation graph that we use throughout our paper.

4 Bounding the Sizes of Bicliques and Independent Sets

In this section, we consider the modulator T of G to bipartite permutation graph obtained in the previous section, and we bound the size of each biclique and independent set in a *complete bipartite decomposition* of $G - T$.

[1] Proofs of results marked with ♣ have been omitted due to space constraints.

Throughout this section, we assume that we have fixed a bipartition (A, B) of $G - T$ and a strong ordering π of (A, B). We also assume that $G - T$ is connected. Later, we will remove this requirement. (We assume connectivity so that we can work with a complete bipartite decomposition of $G - T$.) We also fix a complete bipartite decomposition $\mathcal{D} = (Q_1, R_1, \ldots, Q_s, R_s)$ of $G - T$.

4.1 Auxiliary Results

Next, we prove a few simple results that will be used later to bound the size of each biclique and independent set in the complete bipartite decomposition $\mathcal{D}$ of $G - T$.

▶ **Lemma 9.** *Let H be an induced path in G. Consider $v \in V(G) \setminus V(H)$. If v has more than 5 neighbours in $V(H)$, then $G[V(H) \cup \{v\}]$ contains a small obstruction.*

Proof. Assume that $|N(v) \cap V(H)| \geq 5$. Let H be a path from x to y for some $x, y \in V(G)$. Let $v_1, v_2, v_3, v_4, v_5 \in V(H)$ be the first 5 neighbours of v that appear as we traverse H from x to y. Note that if $v_i v_{i+1} \in E(G)$ for some $i \in [4]$, then, $\{v v_i v_{i+1}\}$ induces a triangle, which is an obstruction, and the lemma follows. So, assume that $v_i v_{i+1} \notin E(H)$ for every $i \in [4]$. This means that no two vertices from $\{v_1, v_2, v_3, v_4, v_5\}$ appear consecutively on H. For $i \in \{1, 3\}$, let u_i be the neighbour of v_i that appears between v_i and v_{i+1} as we traverse H from v_1 to v_5, and let u_5 be the neighbour of v_5 that appears between v_4 and v_5 as we traverse H from v_1 to v_5. Then, notice that $\{v, v_1, u_1, v_3, u_3, v_5, u_5\}$ induces a subdivided claw, which is an obstruction. ◀

▶ **Lemma 10 (♣).** *Let H' be a graph with a Hamiltonian cycle, and let $C = v_1 v_2 \ldots, v_\ell v_1$ be a Hamiltonian cycle in H', where $\ell \geq 45$. Let $Y \subseteq V(H')$ be such that (i) $1 \leq |Y| \leq 3$, (ii) the vertices of Y appear consecutively in the cycle C (i.e., $Y = \{v_i, v_{i+1}, v_{i+2}\}$ for some $i \in [\ell - 2]$ or $Y = \{v_{\ell-1}, v_\ell, v_1\}$ or $Y = \{v_\ell, v_1, v_2\}$), (iii) $H' - Y$ is an induced path and (iv) $d_{H'}(y) \leq 5$ for every $y \in Y$. Then, H' contains an obstruction.*

Proof. Observe first that since $H' - Y$ is an induced path, all the chords in the cycle C are incident with Y. Consider $y \in Y$. Since C is a cycle, $d_C(y) = 2$. And since $d_H(y) \leq 5$, we can conclude that the cycle C has at most 3 chords that are incident with y. Note that if y is adjacent to two vertices that appear consecutively on the cycle C, i.e., if $y v_i, y v_{i+1} \in E(H')$ for some $i \in [\ell - 1]$ or $y v_\ell, y v_1 \in E(H')$, then H' contains a triangle, which is an obstruction. So, assume that there does not exist $y \in Y$ such that y is adjacent to two vertices that appear consecutively on C. Suppose that C does not contain a hole of length at least 5, then for every vertex $v_i \in C$, vertex v_{i+2} is adjacent to a vertex in Y. Intuitively every alternate vertex must have neighbour in Y so that every cycle of length at least 5 have a chord. However, $|N(Y) \cap V(C)| \leq 15$, implies that there is an induced path of length at least 5 such that it does not contain any neighbour of Y. Let P be longest induced path in C such that endpoints of P have neighbours in Y and no internal vertex of P is adjacent to any vertex of Y. Then as there is no triangle in $H'[V(C)]$, we obtain that $V(P)$ together with Y induces a hole of length at least 5, a contradiction. Hence H' contains an obstruction. ◀

4.2 Bounding the Size of a Biclique the Complete Bipartite Decomposition

In this section, we bound the size of each biclique in the complete bipartite decomposition $\mathcal{D} = (Q_1, R_1, \ldots, Q_s, R_s)$ of $G - T$. In particular, we show that if $G - T$ has a sufficiently large biclique, then in polynomial time we can find and safely delete an "irrelevant vertex" from such a biclique. We start with a marking procedure which marks a set of vertices in a given biclique.

Procedure `Mark-1`**.** The procedure works in 3 steps as follows. For each $j \in [s]$, we initialise $M_j = \emptyset$, and do as follows:
Step 1: For each $v \in T$, if $|(N(v) \cap Q_j \cap A) \setminus M_j| \leq k+1$, then we add $(N(v) \cap Q_j \cap A) \setminus M_j$ to M_j, otherwise we add the first $k+1$ vertices in $(N(v) \cap Q_j \cap A) \setminus M_j$ in the ordering π_A to M_j. Similarly, if $|(N(v) \cap Q_j \cap B) \setminus M_j| \leq k+1$, then we add $(N(v) \cap Q_j \cap B) \setminus M_j$ to M_j, else we add the first $k+1$ vertices in $(N(v) \cap Q_j \cap B) \setminus M_j$ in the ordering π_B to M_j.
Step 2: for each $u \in F_{k+1}^{Q_j \cap A} \setminus M_j$, we add u to M_j. And for each $u \in F_{k+1}^{Q_j \cap B} \setminus M_j$, we add u to M_j.
Step 3: for each $u \in L_{k+1}^{Q_j \cap A} \setminus M_j$, we add u to M_j and for each $u \in L_{k+1}^{Q_j \cap B} \setminus M_j$, we add u to M_j.

We now bound the size of the set M_j at the end of the procedure `Mark-1`.

▶ Remark 11. Observe that the Procedure `Mark-1` can be executed in polynomial time. Also note that $|M_j| \leq 2(k+1)(|T|+2)$ for every $j \in [s]$. To see this, fix $j \in [s]$. Note that for each $v \in T$, we add at most $2(k+1)$ vertices to M_j, i.e., at most $k+1$ vertices from $(N(v) \cap Q_j \cap A) \setminus M_j$ and at most $k+1$ vertices from $(N(v) \cap Q_j \cap B) \setminus M_j$. Therefore, the number of vertices we added to M_j in Step 1 is at most $2(k+1)|T|$. And in each of Steps 2 and 3, we add at most $2(k+1)$ vertices to M_j. Thus, $|M_j| \leq 2(k+1)(|T|+2)$.

▶ **Reduction Rule 12.** *If there exists a vertex $v \in Q_j \setminus M_j$ for some $j \in [s]$, then delete v.*

▶ **Lemma 13.** *Reduction Rule 12 is safe.*

Proof. Consider an application of Reduction Rule 12 in which a vertex, say $v \in Q_j \setminus M_j$ was deleted for some $j \in [s]$. We show that (G, k) is a yes-instance of BPVD if and only if $(G - v, k)$ is a yes-instance of BPVD. Observe first that if (G, k) is a yes-instance, then so is $(G - v, k)$, as $G - v$ is an induced subgraph of G. Assume now for a contradiction that $(G - v, k)$ is a yes-instance, but (G, k) is not. And let $X \subseteq V(G - v)$ be a solution of size at most k. That is $(G - v) - X$ is a bipartite permutation graph. And by our assumption that (G, k) is a no-instance, $G - X$ is not a bipartite permutation graph. Then, $G - X$ must contain an obstruction, say, H. Note that $v \in V(H)$, as otherwise, H would be an obstruction in $(G - v) - X$, which contradicts the fact that $(G - v) - X$ is a bipartite permutation graph. We first claim that H is a large obstruction. Suppose not. Note that X hits all obstructions in $G - v$. And since $G[T]$ is a subgraph of $G - v$ as $v \notin T$, X hits all obstructions in the subgraph $G[T]$ as well. In particular, X hits all small obstructions in $G[T]$. Let $Y \subseteq X$ be a minimal hitting set for all small obstructions in $G[T]$. Then, by the definitions of T and Y, we can conclude that Y hits all small obstructions in G as well. But then, as H is an obstruction in $G - X$ and $Y \subseteq X$, we can conlude that H is a small obstruction in $G - Y$, a contradiction. Thus, H is a large obstruction in $G - X$. That is, X is hole of length at least 45.

Let u and w be the neighbours of v in H, i.e., $H = uvw \dots u$. And thus $H - v$ is an induced path from w to u in G. Without loss of generality, assume that $v \in A$. Then, $u, w \in B$. We show that we can construct another hole H' in $(G - v) - X$, which will contradict the fact that $(G - v) - X$ is a bipartite permutation graph. For this, we consider different cases depending on which $Q_i \cup R_i$ or T each of the two vertices u and w belongs to. Recall that $v \in Q_j \setminus M_j$. Notice that for $x \in \{u, w\}$, if $x \notin T$, then, by the definition of a complete bipartite decomposition, $x \in Q_{j-1} \cup Q_j \cup Q_{j+1} \cup R_j$.

1. $u, w \in T$. Notice that in Step 1 of the Procedure `Mark-1`, we have added $k+1$ neighbours of u in $Q_j \cap A$ and $k+1$ neighbours of w in $Q_j \cap A$ to M_j, as otherwise, we would have added v as well to M_j. That is, we have $|M_j(u) \cap A| = k + 1$ and $|M_j(w) \cap A| = k + 1$. Since, $|X| \leq k$, we have $(M_j(u) \cap A) \setminus X \neq \emptyset$ and $(M_j(w) \cap A) \setminus X \neq \emptyset$. Let $u' \in (M_j(u) \cap A) \setminus X$ and $w' \in (M_j(w) \cap A) \setminus X$. Let $v' \in (Q_j \cap B) \setminus (V(H) \cup X)$. Let H' be the graph obtained from H by replacing the vertex v with vertices u', v', w' and edges uv, vw by edges $uu', u'v', u'w', w'w$. Notice that no vertex of H' belongs to $X \cup \{v\}$ and the graph $H' - \{u', v', w'\}$ is an induced path in G. By Lemma 9, each of the vertices u', v' and w' has at most 4 neighbours in $V(H') \setminus \{u', v', w'\}$. By Lemma 10 we conclude that H' contains an obstruction, which is also an obstruction in $(G - v) - X$, contradicts that X is a solution to $G - v$ of BPVD.
2. $u \in T, w \in Q_j \cup R_j \cup Q_{j+1}$. In Step 1 of the Procedure `Mark-1`, we have added $k + 1$ neighbours of u in $Q_j \cap A$ to M_j, as otherwise, we would have added v as well to M_j. Thus, $|M_j(u) \cap A| = k + 1$. And note that $v \notin L_{k+1}^{Q_j \cap A}$, as in Step 3 of the Procedure `Mark-1`, we have also added all the vertices in the set $L_{k+1}^{Q_j \cap A}$. Since $|X| \leq k$, $(M_j(u) \cap A) \setminus X \neq \emptyset$ and $L_{k+1}^{Q_j \cap A} \setminus X \neq \emptyset$. Let $u' \in (M_j(u) \cap A) \setminus X$ and $w' \in L_{k+1}^{Q_j \cap A} \setminus X$. Let $v' \in (Q_j \cap B) \setminus (V(H) \cup X)$. Note that by the definition of u', we have $uu' \in E(G)$. And since Q_j is a biclique, $u'v', v'w' \in E(G)$. If $w \in Q_j$, then $ww' \in E(G)$ as well. If not $w' \in R_j \cup Q_{j+1}$. But then, as $v <_{pi} w', v' <_\pi w$ and $vw, v'w' \in E(G)$, by the definition of the strong ordering, we have $ww' \in E(G)$. Let H' be the graph obtained from H by replacing the vertex v with vertices u', v', w' and edges uv, vw by edges $uu', u'v', u'w', w'w$. Notice that no vertex of H' belongs to $X \cup \{v\}$. Again, the graph $H' - \{u', v', w'\}$ is an induced path in G. And by Lemma 9, each of the vertices u', v' and w' has at most 4 neighbours in $V(H') \setminus \{u', v', w'\}$. By Lemma 10 we conclude that H' contains an obstruction, which is also an obstruction in $(G - v) - X$, contradicts that X is a solution to $G - v$ of BPVD.
3. $u \in T, w \in Q_{j-1}$. In Step 1 of the Procedure `Mark-1`, we must have marked $k+1$ neighbours of u in $Q_j \cap A$, as otherwise, we would have marked v as well. Thus, $|M_j(u) \cap A| = k + 1$. And note that $v \notin F_{k+1}^{Q_j \cap A}$, as in Step 2 of the Procedure `Mark-1`, we have also marked all the vertices in the set $F_{k+1}^{Q_j \cap A}$. Since $|X| \leq k$, $(M_j(u) \cap A) \setminus X \neq \emptyset$ and $F_{k+1}^{Q_j \cap A} \setminus X \neq \emptyset$. Let $u' \in (M_j(u) \cap A) \setminus X$, $w' \in F_{k+1}^{Q_j \cap A} \setminus X$ and $v' \in (Q_j \cap B) \setminus (V(H) \cup X)$. By the definition of u', we have $uu' \in E(G)$. Since Q_j is a biclique, $u'v', v'w' \in E(G)$. And since $w' <_\pi v, w <_\pi v'$ and $wv, w'v' \in E(G)$, by the definition of the strong ordering, we have $ww' \in E(G)$. Let H' be the graph obtained from H by replacing the vertex v with the path $u'v'w'$. Notice that no vertex of H' belongs to $X \cup \{v\}$. Again, the graph $H' - \{u', v', w'\}$ is an induced path in G. And by Lemma 9, each of the vertices u', v' and w' has at most 4 neighbours in $V(H') \setminus \{u', v', w'\}$. By Lemma 10 we conclude that H' contains an obstruction, which is also an obstruction in $(G - v) - X$, contradicts that X is a solution to $G - v$ of BPVD.

The cases where $w \in T, u \in Q_j \cup R_j \sup Q_{j+1}$ and $w \in T, u \in Q_{j-1}$ are symmetric. We have thus covered all the cases in which at least one neighbour of v in H belongs to T. Assume now that $u, w \notin T$. Then, by the definition of complete bipartite decomposition, $u, w \in Q_{j-1} \cup Q_j \cup R_j \cup Q_{j+1}$.

4. $u \in Q_{j-1} \cup Q_j \cup Q_{j+1}$ and $w \in Q_{j-1} \cup Q_j \cup Q_{j+1}$.

Since $|X| \leq k$, $F_{k+1}^{Q_j \cap A} \setminus X \neq \emptyset$ and $L_{k+1}^{Q_j \cap A} \setminus X \neq \emptyset$. If $u \in Q_{j-1} \cup Q_j$, then let $u' \in F_{k+1}^{Q_j \cap A} \setminus X$, otherwise let $u' \in L_{k+1}^{Q_j \cap A} \setminus X$. Similarly, if $w \in Q_{j-1} \cup Q_j$, then let $w' \in F_{k+1}^{Q_j \cap A} \setminus X$, otherwise let $w' \in L_{k+1}^{Q_j \cap A} \setminus X$. Let $v' \in (Q_j \cap B) \setminus (V(H) \cup X)$. (1) If $u \in Q_{j-1}$, then since Q_{j-1} is a biclique, u must have a neighbour u'' in Q_{j-1} and as $u'' <_\pi u' <_\pi v$, i.e., u' is between u'' and v in ordering π, therefore $uu' \in E(G)$ by the definition of strong ordering. (2) If $u \in Q_j$, then since Q_j is a biclique, $uu' \in E(G)$. (3) If $u \in Q_{j+1}$, then since Q_{j+1} is a biclique, u must have a neighbour u'' in Q_{j+1} and as $v <_\pi u' <_\pi u''$, i.e., u' is between u'' and v in ordering π, therefore $uu' \in E(G)$ by the definition of strong ordering. Similar arguments follows for $w \in Q_{j-1} \cup Q_j \cup Q_{j+1}$ and implies that $ww' \in E(G)$. As Q_j is a biclique $uuv', v'w' \in E(G)$. Let H' be the graph obtained from H by replacing the vertex v with the path $u'v'w'$. Notice that no vertex of H' belongs to $X \cup \{v\}$. Again, the graph $H' - \{u', v', w'\}$ is an induced path in G. And by Lemma 9, each of the vertices u', v' and w' has at most 4 neighbours in $V(H') \setminus \{u', v', w'\}$. By Lemma 10 we conclude that H' contains an obstruction, which is also an obstruction in $(G - v) - X$, contradicts that X is a solution to $G - v$ of BPVD.

Assume now that atleast one of u, w in R_j.

5. $u \in Q_{j-1}, w \in R_j$. Note that $v \notin F_{k+1}^{Q_j \cap A} \cup L_{k+1}^{Q_j \cap A}$, as in Steps 3,4 of the Procedure `Mark-1`, we have added all the vertices in the set $F_{k+1}^{Q_j \cap A} \cup L_{k+1}^{Q_j \cap A}$ to M_j. Since $|X| \leq k$, $F_{k+1}^{Q_j \cap A} \setminus X \neq \emptyset$ and $L_{k+1}^{Q_j \cap A} \setminus X \neq \emptyset$. Let $u' \in F_{k+1}^{Q_j \cap A} \setminus X$ and $w' \in L_{k+1}^{Q_j \cap A} \setminus X$. Let $v' \in (Q_j \cap B) \setminus (V(H) \cup X)$. By strong ordering, we have $uu' \in E(G)$. Since Q_j is a biclique, $u'v', v'w' \in E(G)$. As $v <_\pi w'$ and $v' <_\pi w$, we obtain that $vv', ww' \in E(G)$, by the properties of strong ordering. This implies that $uu', u'v', v'w', w'w \in E(G)$. Let H' be the graph obtained from H by replacing the vertex v with vertices u', v', w' and edges uv, vw by $uu', u'v', v'w', w'w$. Notice that no vertex of H' belongs to $X \cup \{v\}$ and H' is a Hamiltonian cycle. Again, the graph $H' - \{u', v', w'\}$ is an induced path in G. And by Lemma 9, each of the vertices u', v' and w' has at most 4 neighbours in $V(H') \setminus \{u', v', w'\}$. By Lemma 10 we conclude that H' contains an obstruction, which is also an obstruction in $(G - v) - X$, contradicts that X is a solution to $G - v$ of BPVD.

The other cases where exactly one of $u, w \in R_j$ are symmetric. Assume now that both $u, w \in R_j$.

6. $u, w \in R_j$. Note that $v \notin L_{k+1}^{Q_j \cap A}$, as in Steps 4 of the Procedure `Mark-1`, we have added all the vertices in the set $L_{k+1}^{Q_j \cap A}$ to M_j. Since $|X| \leq k$, $L_{k+1}^{Q_j \cap A} \setminus X \neq \emptyset$. Let $t \in L_{k+1}^{Q_j \cap A} \setminus X$. Let $v' \in (Q_j \cap B) \setminus (V(H) \cup X)$. Since Q_j is a biclique, $vv', v'w' \in E(G)$. As $v <_\pi t$, $v' <_\pi u$ and $v' <_\pi w$, we obtain that $ut, wt \in E(G)$, by the properties of strong ordering. Let H' be the graph obtained from H by replacing the vertex v with t. and edges uv, vw by ut, tw. Notice that no vertex of H' belongs to $X \cup \{v\}$. Again, the graph $H' - \{t\}$ is an induced path in G. And by Lemma 9, t has at most 4 neighbours in $V(H') \setminus \{t\}$. Thus, $|N(\{t\}) \cap (V(H') \setminus \{t\}| \leq 4$. By Lemma 10 we conclude that H' contains an obstruction, which is also an obstruction in $(G - v) - X$, contradicts that X is a solution to $G - v$ of BPVD. ◀

4.3 Bounding the Size of an Independent Set in the Complete Bipartite Decomposition

In this section we bound the number of vertices in each independent set R_i for each $i \in [s]$ in the complete bipartite decomposition $\mathcal{D}$ of $G - T$. First, we describe construction of a set M_j with respect to an independent set R_j, $j \in [s]$ in the complete bipartite decomposition $\mathcal{D}$ of $G - T$.

Procedure `Mark-2`**.** The procedure works in 4 steps as follows. For each $j \in [s]$, we initialise $M_j = \emptyset$, and do as follows:
Step 1: For each $v \in T$, if $|(N(v) \cap R_j \cap A) \setminus M_j| \leq k+1$, then we add $(N(v) \cap R_j \cap A) \setminus M_j$ to M_j, and otherwise we add the first $k+1$ vertices in $(N(v) \cap R_j \cap A) \setminus M_j$ in the ordering π_A to M_j. Similarly, if $|(N(v) \cap R_j \cap B) \setminus M_j| \leq k+1$, then add $(N(v) \cap R_j \cap B) \setminus M_j$ to M_j, and else we add the first $k+1$ vertices in $(N(v) \cap R_j \cap B) \setminus M_j$ in the ordering π_B to M_j.
Step 2: For each pair $x, y \in T$, if $|(N(x) \cap N(y) \cap R_j \cap A) \setminus M_j| \leq k+1$, then we add $(N(x) \cap N(y) \cap R_j \cap A) \setminus M_j$ to M_j, else we add the first $k+1$ vertices in $(N(x) \cap N(y) \cap R_j \cap A) \setminus M_j$ in the sequence π to M_j. Similarly, for each pair $x, y \in T$, if $|(N(x) \cap N(y) \cap R_j \cap B) \setminus M_j| \leq k+1$, then we add $(N(x) \cap N(y) \cap R_j \cap B) \setminus M_j$ to M_j, else we add first $k+1$ vertices in $(N(x) \cap N(y) \cap R_j \cap B) \setminus M_j$ in the sequence π to M_j.
Step 3: for each $u \in F_{k+1}^{R_j \cap A} \setminus M_j$, we add u to M_j. And for each $u \in F_{k+1}^{R_j \cap B} \setminus M_j$, we add u to M_j.
Step 4: for each $u \in L_{k+1}^{R_j \cap A} \setminus M_j$, we add u to M_j and for each $u \in L_{k+1}^{R_j \cap B} \setminus M_j$, we add u to M_j.

We now bound the size of the set M_j at the end of the procedure `Mark-2`.

▶ Remark 14. Observe that the Procedure `Mark-2` can be executed in polynomial time. Observe also that $|M_j| \leq (k+1)(|T| + |T|^2 + 1)$ for every $j \in [s]$. To see this, fix $j \in [s]$. Note that for each $v \in T$, we added at most $2(k+1)$ neighbours to v to M_j, i.e., at most $2(k+1)$ vertices from $(N(v) \cap R_j) \setminus M_j$. Therefore the number of vertices we added to M_j in Step 1 is at most $2(k+1)|T|$. And in Step 2, for each pair $x, y \in T$, we added at most $2(k+1)$ common neighbours of x and y to M_j, and therefore the number of vertices we added to M_j in Step 2 is at most $2(k+1)|T|^2$. In each of Steps 3 and 4, we added at most $2(k+1)$ vertices to M_j. Thus, $|M_j| \leq 2(k+1)(|T| + |T|^2 + 2)$.

Using the set M_j constructed by Procedure `Mark-2`, we get the following reduction rule.

▶ **Reduction Rule 15.** *If there exists $v \in R_j \setminus M_j$ for some $j \in [s]$, then delete v.*

▶ **Lemma 16.** *Reduction Rule 15 is safe.*

Proof. Consider an application of Reduction Rule 15 in which a vertex, say $v \in R_j \setminus M_j$ was deleted for some $j \in [s]$. We show that (G, k) is a yes-instance of BPVD if and only if $(G - v, k)$ is a yes-instance of BPVD. Observe first that if (G, k) is a yes-instance, then so is $(G - v, k)$, as $G - v$ is an induced subgraph of G. Assume now for a contradiction that $(G - v, k)$ is a yes-instance, but (G, k) is not. And let $X \subseteq V(G - v)$ be a solution of size at most k. That is $(G - v) - X$ is a bipartite permutation graph. And by our assumption that (G, k) is a no-instance, $G - X$ is not a bipartite permutation graph. Then, $G - X$

must contain an obstruction, say, H. Note that $v \in V(H)$, as otherwise, H would be an obstruction in $(G-v)-X$, which contradicts the fact that $(G-v)-X$ is a bipartite permutation graph.

We first claim that H is a large obstruction. Suppose not. Note that X hits all obstructions in $G-v$. And since $G[T]$ is a subgraph of $G-v$ as $v \notin T$, X hits all obstructions in the subgraph $G[T]$ as well. In particular, X hits all small obstructions in $G[T]$. Let $Y \subseteq X$ be a minimal hitting set for all small obstructions in $G[T]$. Then, by the definitions of T and Y, we can conclude that Y hits all small obstructions in G as well. But then, as H is an obstruction in $G-X$ and $Y \subseteq X$, we can conclude that H is a small obstruction in $G-Y$, a contradiction. Thus, H is a large obstruction in $G-X$. That is, H is hole of length at least 45.

Let u and w be the neighbours of v in H, i.e., $H = uvw \dots u$. And thus $H-v$ is an induced path from w to u. Without loss of generality, assume that $v \in A$. Then, $u, w \in B$. We show that we can construct another hole H' in $(G-v)-X$, which will contradict the fact that $(G-v)-X$ is a bipartite permutation graph. For this, we consider different cases depending on which $Q_i \cup R_i$ or T each of the two vertices u and w belongs to. Recall that $v \in R_j \setminus M_j$. Notice that for $x \in \{u, w\}$, if $x \notin T$, then, by the definition of a complete bipartite decomposition, $x \in Q_j$.

1. $u, w \in T$. Notice that as $v \in (N(u) \cap N(w) \cap R_j \cap A) \setminus M_j$, by Step 2 of the Procedure `Mark-2`, we must have marked $k+1$ common neighbours of u, w in $R_j \cap A$, i.e., we have added $k+1$ vertices in $(N(u) \cap N(w) \cap R_j \cap A)$ to M_j as otherwise, we would have added v to M_j as well. That is, we have $|M_j \cap N(u) \cap N(w) \cap A| \geq k+1$. Since, $|X| \leq k$, we have $(M_j \cap N(u) \cap N(w) \cap A) \setminus X \neq \emptyset$. Also notice that $N(u) \cap N(w) \cap V(H) = \{v\}$, as H is a hole. Let $v' \in M_j \cap N(u) \cap N(w) \cap A) \setminus X$ and H' be the graph obtained from H by replacing the vertex v by v' and by replacing edges uv, vw by $uv', v'w$. Notice that no vertex of H' belongs to $X \cup \{v\}$ and the graph $H'-v$ is an induced path in G. And H' is a cycle of length at least 45in G. By Lemma 9, v' have at most 4 neighbours in $H'-v'$. By Lemma 10 we conclude that H' contains an obstruction, which is also an obstruction in $(G-v)-X$, contradicts that X is a solution to $G-v$ of BPVD.
2. $u \in Q_j, w \in T$. (analogous arguments follows for the case $u \in T, w \in Q_j$) In Step 1 of the Procedure `Mark-2`, we have added $k+1$ neighbours of u in $R_j \cap A$ to M_j which are before v in sequence π, as otherwise, we would have added v as well to M_j. Thus, $|N(u) \cap M_j \cap A| = k+1$. Let $v' \in N(w) \cap R_j \cap A \setminus X$. As $v' <_\pi v$, we have $v'u \in E(G)$, by the definition of the strong ordering, as Q_j is a non-trivial biclique and hence u must have a neighbour u' in $Q_j \cap A$ and hence all the vertices between u' to v in π are neighbours of u, which implies $v' \in N(u)$. Let H' be the graph obtained from H by replacing the vertex v with vertex v' and edge uv, vw by edges $uv', v'w$. And by Lemma 9, each of the vertices u', v' and w' has at most 4 neighbours in $V(H') \setminus \{u', v', w'\}$. By Lemma 10 we conclude that H' contains an obstruction, which is also an obstruction in $(G-v)-X$, contradicts that X is a solution to $G-v$ of BPVD.
3. $u, w \in Q_j$. Note that $v \notin F_{k+1}^{R_j \cap A}$, as in Step 3 of the Procedure `Mark-2`, we have added all the vertices in the set $F_{k+1}^{R_j \cap A}$ to M_j. Since $|X| \leq k$, $F_{k+1}^{R_j \cap A} \setminus X \neq \emptyset$. Let $v' \in F_{k+1}^{R_j \cap A} \setminus X$. As $v' <_\pi v$, we have $v'u, v'w \in E(G)$, by the definition of the strong ordering, as Q_j is a non-trivial biclique and hence u, w must have a neighbour u' in $Q_j \cap A$ and hence all the vertices between u' to v in π are neighbours of u, w, which implies $v' \in N(u) \cap N(w)$. Let H' be the graph obtained from H by replacing the vertex v with vertex v' and edge uv, vw by edges $uv', v'w$. And by Lemma 9, each of the vertices u', v' and w' has at most 4 neighbours in $V(H') \setminus \{u', v', w'\}$. By Lemma 10 we conclude that H' contains an obstruction, which is also an obstruction in $(G-v)-X$, contradicts that X is a solution to $G-v$ of BPVD. ◀

▶ **Observation 17.** *After an exhaustive application of Reduction Rule 12, note that for every* $j \in [s]$, $M_j \setminus Q_j = \emptyset$. *Thus, by Remark 11,* $|Q_j| = |M_j| = \mathcal{O}(k \cdot |T|)$.

▶ **Observation 18.** *After an exhaustive application of Reduction Rule 15, note that for every* $j \in [s]$, $M_j \setminus R_j = \emptyset$. *Thus, by Remark 14,* $|R_j| = |M_j| = \mathcal{O}(k \cdot |T|^2)$.

Observation 17 and 18 together imply the following result.

▶ **Lemma 19.** *Given an instance* (G, k) *of* BPVD *and a nice modulator* $T \subseteq V(G)$ *of size* $k^{\mathcal{O}(1)}$, *in polynomial time, we can construct an equivalent instance* (G', k) *such that* G' *is an induced subgraph of* G, $T \subseteq V(G')$, *and for each connected component of* $G' - T$ *with a complete bipartite decomposition* $(Q_1, R_1, \ldots, Q_s, R_s)$, *we have* $|Q_j \cup R_j| = \mathcal{O}(k \cdot |T|^2)$.

5 Bounding the Size of a Connected Component

In this section we bound the size of each connected component in $G - T$. Recall that in previous sections we bounded the size of each Q_i and R_i, $i \in [s]$, in nice decomposition of $G - T$. Our aim in this section is to bound the number of Q_i and R_i in each connected component of $G - T$. Let $\mathcal{C}$ be a connected component in $G - T$. Without loss of generality let $\mathcal{C} = \bigcup_i (Q_i \cup R_i)$. For a pair Q_i, R_i of biclique and independent set, the set $Q_i \cup R_i$ is called a *block*.

▶ **Reduction Rule 20.** *Let* v *be a vertex in* T *such that* v *is contained in at least* $k + 1$ *disjoint triangles* (v, a_i, b_i, v) *intersecting exactly at* $\{v\}$, *where* $a_i, b_i \in V(G) \setminus T$, *then delete* v *from* G, *and reduce* k *by* 1. *The resultant instance is* $(G - v, k - 1)$.

The correctness of above reduction rule is easy to see as every solution to (G, k) of BPVD must contain v. From now onwards we assume that Reduction Rule 20 is not applicable.

▶ **Reduction Rule 21.** *Let* v *be a vertex in* T. *If* v *has more than* $6(k + 1)$ *neighbours* a_i*'s in different* $Q_i \cup R_i$ *such that there exists* $b_i \in N(a_i) \cap Q_i \setminus N(v)$, *then delete* v *from* G, *and reduce* k *by* 1. *The resultant instance is* $(G - v, k - 1)$.

▶ **Lemma 22.** *Reduction Rule 21 is correct.*

Proof. Notice that (v, a_i, b_i) is an induced P_3. By pigeon hole principle there are at least $3(k+1)$ non-consecutive blocks $Q_i \cup R_i$ which contains a pair (a_i, b_i) such that (v, a_i, b_i) is an induced P_3. Let $\mathcal{P}$ be the set of such induced P_3s. That is, $\mathcal{P}$ is a set of distinct induced P_3s, (v, a_i, b_i) intersecting exactly at $\{v\}$ and for every pair of P_3s, (v, a_i, b_i) and (v, a_j, b_j), where $a_i, b_i \in Q_i \cup R_i$ and $a_j, b_j \in Q_j \cup R_j$, the blocks $Q_i \cup R_i$ and $Q_j \cup R_j$ are not consecutive. Notice that we can construct a set of $k + 1$ induced subdivided claws intersecting exactly at v using $\mathcal{P}$, which implies that any solution to (G, k) of BPVD must contain v. ◀

From now onwards we assume that Reduction Rules 20 and 21 are not applicable.

▶ **Lemma 23 (♣).** *Let* $\mathcal{C}$ *be a connected component in* $G - T$. *Then there are at most* $7|T|(k+1)$ *many disjoint blocks* $(Q_i \cup R_i)$ *in nice decomposition of* $\mathcal{C}$ *such that* $N(T) \cap (Q_i \cup R_i) \neq \emptyset$.

If $\mathcal{C}$ has $3500|T|k(k+1)$ disjoint blocks, then by the pigeon hole principle and Lemma 23, there are at least $500k$ consecutive blocks in $\mathcal{C}$ that do not contain any vertex from $N(T)$. Let $Q_1 \cup R_1, \ldots, Q_{500k} \cup R_{500k}$ be the set of $500k$ such consecutive blocks in $\mathcal{C}$ that are disjoint from $N(T)$. Let $j = 500k/2$. Consider $\mathcal{D}_L = \{Q_i \cup R_i | i \in [j - 2k, j - 3]\} \setminus R_{j-3}$ and $\mathcal{D}_R = \{Q_i \cup R_i | i \in [j + 3, j + 2k]\}$. Let $F = \{R_{j-3}\} \cup \{Q_i \cup R_i | i \in [j - 2, j + 2]\}$

and $Z = \{Q_i | i \in [j-2k, j+2k]\}$. Observe that, for a vertex $v \in \mathcal{D}_L \cup \mathcal{D}_R$ and a vertex $u \in T$, $\text{dist}_G(u,v) \geq 240k$. This observation will be used in in proving further results. Let $Q = Q_{j-3}$ and $Q' = Q_{j+3}$. Let Y be a $Q_i, Q_{i'}$ cut in $G-T$, where $i \in [j-2k, j-3]$ and $i' \in [j+3, j+2k]$, where Y must contain vertices from only block $Q_a \cup R_a$, $a \in [i+1, i'-1]$. Let τ be the size of minimum $Q_i, Q_{i'}$ cut in $G-T$ over all pairs i, i', $i \in [j-2k, j-3]$ and $i' \in [j+3, j+2k]$.

▶ **Reduction Rule 24.** *Let F be as defined above. Delete all the vertices of F from G. Introduce three new bicliques $S_1 = K_{k^2,k^2}, S_2 = K_{\lceil \tau/2 \rceil, \lfloor \tau/2 \rfloor}, S_3 = K_{k^2,k^2}$. Also add edges such that $G[V(Q) \cup S_1]$ and $G[S_1 \cup S_2]$, $G[S_2 \cup S_3]$ and $G[V(Q') \cup S_3]$ are complete bipartite graphs. The bicliques appear in the order Q, S_1, S_2, S_3, Q'.*

Let G' be the reduced graph after an application of Reduction Rule 24. Let $S = S_1 \cup S_2 \cup S_3$. Notice that $G' - T$ is a bipartite permutation graph by construction.

▶ **Observation 25 (♣).** *There are no small obstructions containing any vertices from $F \cup \mathcal{D}_L \cup \mathcal{D}_R$ in G. There are no small obstructions containing any vertices from $S \cup \mathcal{D}_L \cup \mathcal{D}_R$ in G'.*

▶ **Observation 26.** *Any hole H in G which contains a vertex from $F \cup \mathcal{D}_L \cup \mathcal{D}_R$, intersects all bicliques in $F \cup \mathcal{D}_L \cup \mathcal{D}_R$. And such H is of length at least $500k$.*

Proof. Since there are no *large* holes in $G-T$, $V(H) \cap T \neq \emptyset$. Without loss of generality, suppose that H intersects a block $Q_i \cup R_i$ but does not intersect some $Q_{i+1} \in Z$. Then any biclique $Q_{i'}$ where $i' < i$ contains at least two vertices from the hole H. Let a_1 and a_2 be two such vertices such that they have an induced path between them in H. Let $H = (s, v_1, v_2, ...a_1, ...a_2, ..., s)$. Notice that a_1 and a_2 can not belong to different partitions of Q_{i-21} since H is a hole. But Q_{i-21} has some vertex v in its other partition. But then we get a cycle $C = (s, ..., a_1, v, a_2, ..., s)$. But v can have at most 5 neighbors on the induced path of the hole $(a_1, ..., s, ..., a_2)$, otherwise there is a small obstruction containing v which is completely contained in $G-T$ which is not possible. Since cycle C has at length at least 40, we can construct a new hole H_1 such that $V(H_1) \subseteq V(C)$ which is completely contained in $G-T$, which is a contradiction. Notice that any such hole must have one vertex from each of the $500k$ consecutive bicliques. Hence the hole has length more than $500k$. ◀

The following claim can be argued similarly.

▶ **Observation 27.** *Any hole H in G' which contains a vertex from $S \cup \mathcal{D}_L \cup \mathcal{D}_\mathcal{R}$, intersects all the bicliques in $S \cup \mathcal{D}_L \cup \mathcal{D}_R$.*

▶ **Lemma 28 (♣).** *Reduction Rule 24 is safe.*

With the above reduction rule we obtain the following result.

▶ **Lemma 29.** *Given an instance (G,k) of* BPVD *and a nice modulator $T \subseteq V(G)$ of size $k^{\mathcal{O}(1)}$, in polynomial time, we can construct an equivalent instance (G',k) such that, $T \subseteq V(G')$, T is a nice modulator for G' and for each connected component $\mathcal{C}$ of $G'-T$ with a complete bipartite decomposition $(Q_1, R_1, \ldots, Q_s, R_s)$, the number of blocks $(Q_i \cup R_i)$s in the connected component $\mathcal{C}$ is at most $3500|T|k^2 = \mathcal{O}(k^2 \cdot |T|)$.*

6 Bounding the Number of Connected Components

Until now we have assumed that $G - T$ is connected. Further, in Section 5, we showed that the size of any connected component is upper bounded by $k^{\mathcal{O}(1)}$. In this section we show that the number of connected components in $G - T$ is also upper bounded by $k^{\mathcal{O}(1)}$. This together with the fact that $|T| \leq k^{\mathcal{O}(1)}$, result in a polynomial kernel for BPVD.

A connected component that has no neighbor in T is a bipartite permutation graph. Hence, we can safely remove it from our instance.

▶ **Reduction Rule 30.** *If there is a connected component $\mathcal{C}$ in $G-T$ such that $N(T) \cap V(\mathcal{C}) = \emptyset$, then reduce (G, k) to $(G - V(\mathcal{C}), k)$.*

From now onwards, we assume that the Reduction Rules 20, 21 and 30 are not applicable. We partition the set of all the connected components in $G - T$ into two sets $\mathbb{C}_{\geq 2}$ and $\mathbb{C}_{=1}$, where $\mathbb{C}_{\geq 2}$ contains all the connected components of size at least 2 whereas, $\mathbb{C}_{=1}$ contains all the connected components of size exactly 1. First, we bound the size of $\mathbb{C}_{\geq 2}$.

▶ **Lemma 31.** $|\mathbb{C}_{\geq 2}| \leq 7|T|(k+1)$.

Proof. Consider any vertex $v \in T$ such that v has a neighbor, say, a, in a connected component, say, $\mathcal{C}_i$, where $\mathcal{C}_i \in \mathbb{C}_{\geq 2}$. Note that for vertex a_i, there exists a neighbor $b_i \in \mathcal{C}_i$ since $\mathcal{C}_i$ has size at least 2.

Case 1: (The vertex b_i is adjacent to v.) Therefore, we have a triangle (v, a_i, b_i, v). If v has more than $k+1$ such different pairs of (a_i, b_i) such that b_i is adjacent to v, then there are $k+1$ triangles of the form (v, a_i, b_i, v) having a common vertex v. It implies that any solution of size k must contain v. By non-applicability of Reduction Rule 20 such case cannot occur. Hence, for any vertex $v \in T$, v has neighbors (a_i's) in at most $k+1$ different components $\mathcal{C}_i \in \mathbb{C}_{\geq 2}$ such that there is a vertex $b_i \in \mathcal{C}_i \cap N(v)$.

Case 2: (The vertex b_i is not adjacent to v.) Therefore, (v, a_i, b_i) is an induced P_3. Let v has more than $6(k+1)$ neighbors (a_i's) in different $\mathcal{C}_i$ such that there exists $b_i \in \mathcal{C}_i \setminus N(v)$. Therefore, there exists some $Q_i \cup R_i$ in component $\mathcal{C}_i$ such that $a_i \in Q_i \cup R_i$ and $b_i \in N(a_i) \cap Q_i \setminus N(v)$. Since vertex v has more that $6(k+1)$ such neighbors a_i, Reduction Rule 21 would be applicable. By non-applicability of Reduction Rule 21 such case cannot occur. Hence, for any vertex $v \in T$, v has neighbors (a_i's) in at most $6(k+1)$ different components $\mathcal{C}_i$ such that there is a vertex $b_i \in N(a_i) \cap Q_i \setminus N(v)$.

Thus, every vertex $v \in T$ has neighbors at most in $(k+1) + 6(k+1)$, that is, $7(k+1)$ different components $\mathcal{C}_i$'s. Hence, $|\mathbb{C}_{\geq 2}| \leq 7|T|(k+1)$. ◀

Next, we proceed to bound the size of the set $\mathbb{C}_{=1}$. Towards that we will utilize the next marking scheme.

Procedure `Mark-3`. We initialise $M = \emptyset$ and for each $\{x, y\} \subseteq T$, we initialise $M(x, y) = \emptyset$, and do as follows: For each $\{x, y\} \subseteq T$, if $|M(x, y)| \leq k+1$ and if there exists $u \in \mathbb{C}_{=1}$ such that $u \in (N(x) \cap N(y)) \setminus M$, then we add u to $M(x, y)$ and M, i.e., we set $M(x, y) \leftarrow M(x, y) \cup \{u\}$ and $M \leftarrow M \cup \{u\}$.

▶ Remark 32. Observe first that $M = \bigcup_{\{x,y\} \subseteq T} M(x, y)$. And in the procedure `Mark-3`, corresponding to each $\{x, y\} \subseteq T$, we add at most $k+1$ vertices to $M(x, y)$. Thus, $|M(x, y)| \leq k+1$, and therefore, $|M| \leq (k+1)\binom{|T|}{2}$, as there are $\binom{|T|}{2}$ many distinct sets $\{x, y\} \subseteq T$.

▶ **Reduction Rule 33.** *If there exists $v \in \mathbb{C}_{=1} \setminus M$, then delete v.*

▶ **Lemma 34 (♣).** *Reduction Rule 33 is safe.*

Observe that by Remark 32 and by applying the Reduction Rule 33 repeatedly, we can reduce the graph such that in the reduced instance, $|\mathbb{C}_{=1}| \leq (k+1)\binom{|T|}{2}$. This reduction and Lemma 31 implies the following result:

▶ **Lemma 35.** *Given an instance (G,k) and a nice modulator $T \subseteq V(G)$ of size $k^{\mathcal{O}(1)}$, in polynomial time, we can construct an equivalent instance (G',k) such that the number of connected component in $G'-T$ is $\mathcal{O}(k \cdot |T|^2)$.*

7 Kernel size analysis

Now we are ready to prove the main result of our paper, that is, Theorem 1. Before proceeding with the proof, let us state all the bounds that contributes to the kernel size.

Size of nice modulator T: $\mathcal{O}(k^{45})$
Number of connected components in $G-T$: $\mathcal{O}(k \cdot |T|^2)$.
Number of blocks in any connected component in $G-T$: $\mathcal{O}(k^2 \cdot |T|)$
Size of any block $(Q_i \cup R_i)$ in $G-T$: $\mathcal{O}(k \cdot |T|^2)$.

Proof of Theorem 1. Let (G,k) be an instance to the BPVD problem. First we show that if G is not connected we can reduce it to the connected case. If there is a connected component $\mathcal{C}$ that is a bipartite permutation graph, we delete it. Clearly, (G,k) is a yes instance if and only if $(G \setminus \mathcal{C}, k)$ is a yes instance. We repeat this process until every connected component of G is not a bipartite permutation graph. At this stage if the number of connected components is at least $k+1$, then we conclude that G can not be made into a bipartite permutation graph by deleting at most k vertices. Thus, we assume that G has at most k connected components. Now we show how to obtain a kernel for the case when G is connected, and for the disconnected case, we just run this algorithm on each connected component. This only increases the kernel size by a factor of k. From now onwards we assume that G is connected.

From Lemma 8, in polynomial time, we can obtain a nice modulator $T \subseteq V(G)$ of size $\mathcal{O}(k^{45})$ or concludes that (G,k) is a no-instance.

Note that, $G-T$ is a bipartite permutation graph. Next, we take the complete bipartite decomposition of each component in $G-T$. Now by Theorem 35, in polynomial time we return a graph G such that $G-T$ has $\mathcal{O}(k \cdot |T|^2)$ components.

Next, we show how to obtain a kernel for one connected component in $G-T$ and we just run this algorithm on each connected component. This only increases the kernel size by a factor of $\mathcal{O}(k \cdot |T|^2)$. From now onwards we assume that G' is a connected component in $G-T$. By Lemma 29, in polynomial time we can reduce the graph G' such that G' has at most $\mathcal{O}(k^2 \cdot |T|)$ blocks $Q_i \cup R_i$. Next, we bound the size of each block $Q_i \cup R_i$ in G'. By Lemma 19, in polynomial time we can reduce the graph G' such that for each block $Q_i \cup R_i$, $|Q_j \cup R_j| = \mathcal{O}(k \cdot |T|^2)$. Therefore the total number of vertices in any connected component G is at most $\mathcal{O}(k \cdot |T|^2) \cdot \mathcal{O}(k^2 \cdot |T|)$, that is, $\mathcal{O}(k^3 \cdot |T|^3)$.

As the graph $G-T$ has at most $\mathcal{O}(k \cdot |T|^2)$ number of components, the total size of the graph $G-T$ is at most $\mathcal{O}(k \cdot |T|^2) \cdot \mathcal{O}(k^3 \cdot |T|^3)$, that is, $\mathcal{O}(k^4 \cdot |T|^5)$. It follows that $|V(G)| = \mathcal{O}(k^4 \cdot |T|^5) + |T|$, that is, $\mathcal{O}(k^4 \cdot |T|^5)$. Recall that $|T| = \mathcal{O}(k^{45})$. Therefore, the size of the obtained kernel is $\mathcal{O}(k^4 \cdot |T|^5)$, that is, $\mathcal{O}(k^{229})$. ◀

8 Conclusion

In this paper we studied Bipartite Permutation Vertex Deletion from the perspective of kernelization complexity, and designed a polynomial kernel of size $\mathcal{O}(k^{229})$. This answers an open question posed by Bożyk et al. [4]. We remark that the size of kernel can be brought closer to $\mathcal{O}(k^{100})$ by doing more careful case analysis. However, getting a kernel of size $\mathcal{O}(k^{20})$ would require significantly new ideas and we leave that as an open problem. Indeed, showing whether Permutation Vertex Deletion is FPT remains a challenging open problem.

References

1 Akanksha Agrawal, Daniel Lokshtanov, Pranabendu Misra, Saket Saurabh, and Meirav Zehavi. Feedback vertex set inspired kernel for chordal vertex deletion. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1383–1398, 2017.

2 Akanksha Agrawal, Pranabendu Misra, Saket Saurabh, and Meirav Zehavi. Interval vertex deletion admits a polynomial kernel. 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 1711–1730. SIAM, 2019.

3 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *Journal of Computer and System Sciences*, 75(8):423–434, 2009.

4 Lukasz Bozyk, Jan Derbisz, Tomasz Krawczyk, Jana Novotná, and Karolina Okrasa. Vertex deletion into bipartite permutation graphs. In Yixin Cao and Marcin Pilipczuk, editors, *15th International Symposium on Parameterized and Exact Computation, IPEC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 180 of *LIPIcs*, pages 5:1–5:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

5 Andreas Brandstädt, Van Bang Le, and Jeremy P Spinrad. *Graph classes: a survey*. SIAM, 1999.

6 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer-Verlag, 2015.

7 Holger Dell and Dániel Marx. Kernelization of packing problems. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 68–81, 2012.

8 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.

9 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

10 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

11 Andrew Drucker. New limits to classical and quantum instance compression. *SIAM Journal on Computing*, 44(5):1443–1479, 2015.

12 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer-Verlag, Berlin, 2006.

13 F. V. Fomin, S. Saurabh, and Y. Villanger. A Polynomial Kernel for Proper Interval Vertex Deletion. *SIAM Journal on Discrete Mathematics*, 27(4):1964–1976, 2013.

14 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar f-deletion: Approximation, kernelization and optimal FPT algorithms. In *53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 470–479, 2012.

15 Fedor V. Fomin and Saket Saurabh. Kernelization methods for fixed-parameter tractability. In *Tractability*, pages 260–282. Cambridge Univ. Press, Cambridge, 2014.

16 Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *Journal of Computer and System Sciences*, 77(1):91–106, 2011.

17 Toshihiro Fujito. A unified approximation algorithm for node-deletion problems. *Discrete Applied Mathematics*, 86:213–231, 1998.

18 Martin Charles Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, 1980.

19 Jiong Guo and Rolf Niedermeier. Invitation to data reduction and problem kernelization. *SIGACT News*, 38(1):31–45, 2007.

20 Danny Hermelin, Stefan Kratsch, Karolina Soltys, Magnus Wahlström, and Xi Wu. A completeness theory for polynomial (turing) kernelization. *Algorithmica*, 71(3):702–730, 2015.

21 Danny Hermelin and Xi Wu. Weak compositions and their applications to polynomial lower bounds for kernelization. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 104–113, 2012.

22 Bart M. P. Jansen and Marcin Pilipczuk. Approximation and kernelization for chordal vertex deletion. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1399–1418, 2017.

23 Yuping Ke, Yixin Cao, Xiating Ouyang, Wenjun Li, and Jianxin Wang. Unit interval vertex deletion: Fewer vertices are relevant. *J. Comput. Syst. Sci.*, 95:109–121, 2018.

24 Stefan Kratsch. Recent developments in kernelization: A survey. *Bulletin of the EATCS*, 113, 2014.

25 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.

26 Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Kernelization - preprocessing with a guarantee. In *The Multivariate Algorithmic Revolution and Beyond - Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*, pages 129–161, 2012.

27 Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM*, 41:960–981, 1994.

28 Dániel Marx. Chordal deletion is fixed-parameter tractable. *Algorithmica*, 57(4):747–768, 2010.

29 Ross M. McConnell and Jeremy P. Spinrad. Modular decomposition and transitive orientation. *Discret. Math.*, 201(1-3):189–241, 1999.

30 Rolf Niedermeier. *Invitation to fixed-parameter algorithms*, volume 31. Oxford University Press, 2006.

31 Jeremy P. Spinrad, Andreas Brandstädt, and Lorna Stewart. Bipartite permutation graphs. *Discret. Appl. Math.*, 18(3):279–292, 1987.

32 Ryuhei Uehara and Gabriel Valiente. Linear structure of bipartite permutation graphs and the longest path problem. *Inf. Process. Lett.*, 103(2):71–77, 2007.

33 Mihalis Yannakakis. Node- and edge-deletion np-complete problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC)*, pages 253–264, 1978.

Hardness of Metric Dimension in Graphs of Constant Treewidth

Shaohua Li
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Marcin Pilipczuk
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Abstract

The Metric Dimension problem asks for a minimum-sized *resolving set* in a given (unweighted, undirected) graph G. Here, a set $S \subseteq V(G)$ is *resolving* if no two distinct vertices of G have the same distance vector to S. The complexity of Metric Dimension in graphs of bounded treewidth remained elusive in the past years. Recently, Bonnet and Purohit [IPEC 2019] showed that the problem is W[1]-hard under treewidth parameterization. In this work, we strengthen their lower bound to show that Metric Dimension is NP-hard in graphs of treewidth 24.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Graph algorithms, parameterized complexity, width parameters, NP-hard

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.24

Related Version *Full Version*: https://arxiv.org/abs/2102.09791

Funding This research is part of a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, grant agreement 714704.

1 Introduction

Let G be an unweighted and undirected graph and let $S \subseteq V(G)$. For a vertex $v \in V(G)$, the *distance vector* from v to S is the assignment $S \ni w \mapsto \mathrm{dist}_G(v, w)$, where dist_G denotes the distance in the graph G. The set S is *resolving* if a distance vector to S uniquely determines the source vertex; that is, no two vertices of G have the same distance vector to S. The Metric Dimension problem asks for a resolving set of minimum possible size; such a set is sometimes called the *metric basis* of G. The decision version of Metric Dimension asks for a resolving set of size not exceeding a given threshold k.

Metric Dimension has already been introduced in 1970s [7, 12]. Determining its computational complexity turned out to be quite challenging. It is polynomial-time solvable on trees [7, 12, 10], outerplanar graphs [3], and chain graphs [6], but NP-hard for example on planar graphs [3] or split graphs [5]. From the parameterized complexity point of view, the FPT status of the Metric Dimension parameterized by the solution size has been open for a while and finally resolved in negative by Hartung and Nichterlein [8]. In the search of a tractable structural parameterization, FPT algorithms has been shown for parameters: treelength plus maximum degree [1], vertex cover number [8], max leaf number [4], and modular-width [1].

The above list misses probably the most important graph width measure, namely treewidth. Determining the complexity of Metric Dimension, parameterized by treewidth, remained elusive in the last years, and has been asked a few times [1, 3, 4]. Bonnet and Purohit in a paper presented at IPEC 2019 [2] showed that the problem is W[1]-hard, even with pathwidth parameterization. In this work we strengthened their result by proving para-NP-hardness of this parameterization.

▶ **Theorem 1.** *Metric Dimension, restricted to graphs of treewidth at most* 24*, is NP-hard.*

Theorem 1 brings us much closer to closing (unfortunately mostly in negative) the question of the complexity of Metric Dimension in graphs of bounded treewidth. The remaining gap is to determine the exact treewidth value where the problem becomes hard: note that it is open if Metric Dimension is polynomial-time solvable on graphs of treewidth 2.

The proof of Theorem 1 starts with a construction of a graph with a separation of order 9 over which *a lot* of information on a partial solution to Metric Dimension is transfered. More formally, similarly as Bonnet and Purohit [2], we use the Multicolored Resolving Set problem as an auxiliary intermediate problem. In this problem, the input graph is additionally equipped with an integer k, a tuple of k disjoint vertex sets $X_1, X_2, \ldots, X_k$, and a set $\mathcal{P}$ of vertex pairs. The goal is to choose a set S consisting of exactly one vertex from each set X_i so that for every $\{u, v\} \in \mathcal{P}$, the pair $\{u, v\}$ is resolved by S, that is, u and v have different distance vectors to S. In our construction, the sets X_i are on one side of the said separation of order 9, while the pairs $\mathcal{P}$ are on the second side. The crux of the construction is to make every distance from a vertex of the separator to a chosen vertex of S count: despite the fact that the separation has constant size, S is of unbounded size, giving $\Omega(|S|)$ distances to work with. Overall, the above gives a relatively clean reduction giving NP-hardness of Multicolored Resolving Set in graphs of constant treewidth, presented in Section 3. This reduction is the main new insight and technical contribution of this paper.

Then, again similarly as in the work of Bonnet and Purohit [2], it takes a lot of effort (presented in Section 4) to turn the above reduction to Multicolored Resolving Set into a reduction to Metric Dimension. Here, there are two problems. First, one needs to introduce some gadgets to force the solution to take exactly one vertex from each set X_i. Second, one needs to ensure that the intended solution resolves *all* vertex pairs, not only the ones from $\mathcal{P}$. For both problems, we borrow the tools from Bonnet and Purohit [2]. In particular, the first problem is resolved by *forced set gadgets* in a way very similar to [2]. The second problem is resolved by adding a number of new connections to the graph and *forced vertex gadgets* of [2]. Thus, while the toolbox remains the same as in [2], the application is different as the graph we are working with is significantly different. The construction is presented in Sections 4.1-4.2.

After applying all the modifications to obtain a Metric Dimension instance, one needs to check three aspects. First, one needs to ensure that the forced set gadgets work as intended, forcing the solution to take one vertex from each X_i; this check is rather simple and very similar to the analogous check of [2]. Second, one needs to check that the introduced forced vertex gadgets, that contain extra vertices from the intended resolving set (apart from the ones in X_is), do not accidentally resolve any pair from $\mathcal{P}$. This check is not trivial, but still relatively simple. Note that the mentioned two properties already ensure one of the implications in the proof of the correctness of the reduction: if the final Metric Dimension instance is a yes-instance, then the input instance of the source problem is a yes-instance. These two checks are presented in Section 4.3.

Then one needs to check that every pair of vertices is resolved by an intended solution. Due to the complexity of the construction and the properties of this problem, this turned out to be long and arduous (the full proof is deferred to the full version of the paper).

Besides, we show that the treewidth of the constructed graph is bounded by a constant in Section 4.4.

2 Preliminaries

In this paper, all graphs are undirected. In a graph G, let $V(G)$ be the set of vertices of G. For a vertex $v \in V(G)$, we denote the open neighborhood and closed neighborhood of v by $N_G(v)$ and $N_G[v]$ respectively (or just $N(v)$ and $N[v]$ if the graph is clear in the context). For two vertices $u, v \in V(G)$, let $P(u, v)$ be a path from u to v. Since the graph is undirected, $P(u, v)$ and $P(v, u)$ denote the same path. We denote the neighbor of u on $P(u, v)$ by $N_u(u, v)$ (also the neighbor of v on $P(u, v)$ by $N_v(u, v)$). Similarly, if there is a path which is named as, for example, $P^h(i, j, x)$ such that u is one endpoint of $P^h(i, j, x)$, we denote the neighbor of u on $P^h(i, j, x)$ by $N^h_u(i, j, x)$. For simplicity, we abuse the notation in the sense that for a path P, we use P to denote the path or the vertex set of the path. The meaning should be clear in the context. We define the length of a path P to be the number of edges on the path and denote it by $|P|$. For two vertices $u, v \in V(G)$, we define the distance between u and v to be the length of any shortest path from u to v, denoted by $\text{dist}_G(u, v)$. Note that in this paper we use $\text{dist}(u, v)$ to denote the distance between u and v mostly if the graph is clear in the context. For a path P of even length with two endpoints u and v, let w be the vertex on P such that the length of the subpath of P from u to w equals the length of the subpath of P from w to v. Then we call w the *middle vertex* of P and denote it by $\text{mid}(P)$. We say that two distinct vertices u, u' are *false twins* if $N[u] = N[u']$. Since a path decomposition is also a tree decomposition, the treewidth of a graph G is at most the pathwidth of G. In this paper, for convenience of the proof, we use the alternative characterization of pathwidth, i.e. the pathwidth of a graph G equals the *node search number* of G minus one [11]. The definition of the node search number comes from the node search game. We give an informal definition of the node search game as follows. Imagine that the edges of an undirected graph G are tunnels and they are contaminated by some gas. We need to put searchers on vertices of G to clean the gas. The rule is that when the two endpoints of an edge are occupied by two searchers, this edge becomes clean. However, if we remove some searchers from the graph, a cleaned edge can be recontaminated immediately through an unoccupied endpoint to which a contaminated edge is incident. The node search number of G is the minimum number of searchers required to clean all edges of G.

3 Reduction from 3-Dimensional Matching to Multicolored Resolving Set

Bonnet and Purohit introduced k-Multicolored Resolving Set as an intermediate problem in order to show the W[1]-hardness of Metric Dimension parameterized by treewidth [2].

k-Multicolored Resolving Set

Input: An undirected graph $G = (V, E)$, an integer k, a set $\chi = \{X_1, ..., X_k\}$ where $X_1, ..., X_k$ are disjoint subsets of $V(G)$ and a set $\mathcal{P} = \{\{x_1, y_1\}, ..., \{x_h, y_h\}\}$ where $\{x_1, y_1\}, ..., \{x_h, y_h\}$ are vertex pairs of G.

Question: Is there a set of k vertices S such that
(i) $|S \cap X_i| = 1$ for every $i = 1, ..., k$, and
(ii) for every $\ell \in \{1, ..., h\}$, there exists a vertex $v \in S$ such that $\text{dist}(v,x_\ell) \neq \text{dist}(v,y_\ell)$.

We show that this problem is NP-hard on graphs of constant treewidth. We make a reduction from 3-dimensional matching, which is well-known to be NP-hard [9].

3-DIMENSIONAL MATCHING

Input: the universe $U = \{1, 2, 3\} \times [n]$ and a set $\mathcal{F} = \{A_1, ..., A_m\}$ of tuples such that for every $j \in [m]$, the tuple $A_j = \{(1, x), (2, y), (3, z)\}$ where $(1, x), (2, y), (3, z) \in U$.

Question: are there n tuples $A_{j_1}, ..., A_{j_n}$ such that $\bigcup_{h=1}^{n} A_{j_h} = U$.

Given an instance $(U, \mathcal{F})$ of 3-DIMENSIONAL MATCHING with the universe $U = \{1, 2, 3\} \times [n]$ and a set $\mathcal{F}$ of m tuples $A_1, ..., A_m \subseteq U$, we construct an instance $(G, n, \chi, \mathcal{P})$ of n-MULTICOLORED RESOLVING SET as follows. First, we create m vertices $s_i^1, ..., s_i^m$ as X_i for each $i \in [n]$. Let $\chi = \{X_1, ..., X_n\}$ and $X = \bigcup_{i=1}^{n} X_i$. Then we create n vertex pairs $\{u_r^1, v_r^1\}, ..., \{u_r^n, v_r^n\}$ for each $r \in \{1, 2, 3\}$ and let $\mathcal{P}_r = \{\{u_r^i, v_r^i\} | i = 1, ..., n\}$. We create 3 vertices a_r, b_r, c_r and let $W_r = \{a_r, b_r, c_r\}$ for each $r \in \{1, 2, 3\}$. Let $\mathcal{P} = \mathcal{P}_1 \cup \mathcal{P}_2 \cup \mathcal{P}_3$ and $W = W_1 \cup W_2 \cup W_3$. Finally, let $M = 40(n + 1)$. For each tuple $A_j = \{(1, x), (2, y), (3, z)\}$ $(j \in [m], x, y, z \in [n])$ of the given instance and each integer $i \in [n]$, we link s_i^j to a_1, b_1, c_1 with three paths $P(s_i^j, a_1), P(s_i^j, b_1), P(s_i^j, c_1)$ of lengths $\frac{M}{2} + 10x$, $\frac{M}{2} + 5x + 1$ and $\frac{M}{2} - 10x$ respectively, link s_i^j to a_2, b_2, c_2 with three paths $P(s_i^j, a_2), P(s_i^j, b_2), P(s_i^j, c_2)$ of lengths $\frac{M}{2} + 10y$, $\frac{M}{2} + 5y + 1$ and $\frac{M}{2} - 10y$ respectively, and link s_i^j to a_3, b_3, c_3 with three paths $P(s_i^j, a_3), P(s_i^j, b_3), P(s_i^j, c_3)$ of lengths $\frac{M}{2} + 10z$, $\frac{M}{2} + 5z + 1$ and $\frac{M}{2} - 10z$ respectively. For every vertex pair $\{u_r^p, v_r^p\}$ $(p \in [n], r \in \{1, 2, 3\})$, we link u_r^p to a_r, b_r, c_r with three paths $P(u_r^p, a_r), P(u_r^p, b_r), P(u_r^p, c_r)$ of lengths $\frac{M}{2} - 10p$, $\frac{M}{2} - 5p - 1$ and $\frac{M}{2} + 10p$ respectively, and link v_r^p to a_r, b_r, c_r with three paths $P(v_r^p, a_r), P(v_r^p, b_r), P(v_r^p, c_r)$ of lengths $\frac{M}{2} - 10p$, $\frac{M}{2} - 5p - 2$ and $\frac{M}{2} + 10p$ respectively. This finishes the construction. See Figure 1 for an example.

▶ **Lemma 2.** *For an arbitrary vertex pair $\{u_r^x, v_r^x\} \in \mathcal{P}$ $(r \in \{1, 2, 3\}, x \in [n])$,$\{u_r^x, v_r^x\}$ is resolved by s_i^j $(i \in [n], j \in [m])$ if and only if $(r, x) \in A_j$.*

Proof. On one hand, suppose that $(r, x) \in A_j$. For an arbitrary $i \in [n]$, the three paths from s_i^j to u_r^x via a_r, b_r and c_r have lengths M, M and M respectively. The three paths from s_i^j to v_r^x via a_r, b_r and c_r have lengths $M, M - 1$ and M respectively. Note that there could be other paths from s_i^j to v_r^x or u_r^x that go repeatedly between vertices in X and vertices in W. However, the lengths of such paths are at least $M - 20n + M - 10n > M$. As a result, the shortest paths from s_i^j to u_r^x and v_r^x are of lengths M and $M - 1$ respectively. Thus $\{u_r^x, v_r^x\}$ is resolved by s_i^j.

On the other hand, for an arbitrary tuple $A_i = \{(1, p_1), (2, p_2), (3, p_3)\}$, the paths from the vertex s_i^j $(i \in [n])$ to u_r^x $(r \in \{1, 2, 3\})$ via a_r, b_r and c_r have lengths $M + 10(p_r - x), M + 5(p_r - x)$ and $M - 10(p_r - x)$ respectively. The paths from the vertex s_i^j $(i \in [n])$ to v_r^x $(r \in \{1, 2, 3\})$ via a_r, b_r and c_r have lengths $M + 10(p_r - x), M + 5(p_r - x) - 1$ and $M - 10(p_r - x)$ respectively. Note that the paths from s_i^j to u_r^x (or v_r^x) that go repeatedly between the vertices in X and the vertices in W have lengths at least $M - 20n + M - 10n > M + 10n$. They are not the shortest paths from s_i^j to u_r^x (or v_r^x). If $p_r < x$, the shortest paths from s_i^j to u_r^x and v_r^x both have lengths $M + 10(p_r - x)$. If $p_r > x$, the shortest paths from s_i^j to u_r^x and v_r^x both have lengths $M - 10(p_r - x)$. If $p_r = x$, the shortest paths from s_i^j to u_r^x and v_r^x have lengths M and $M - 1$ respectively. As a result, if $\{u_r^x, v_r^x\}$ is resolved by s_i^j, then $p_r = x$. According to the construction, $(r, x) \in A_j$. ◀

▶ **Lemma 3.** *The constructed instance $(G, n, \chi, \mathcal{P})$ of n-MULTICOLORED RESOLVING SET is a yes-instance if and only if the given instance $(U, \mathcal{F})$ of 3-DIMENSIONAL MATCHING is a yes-instance.*

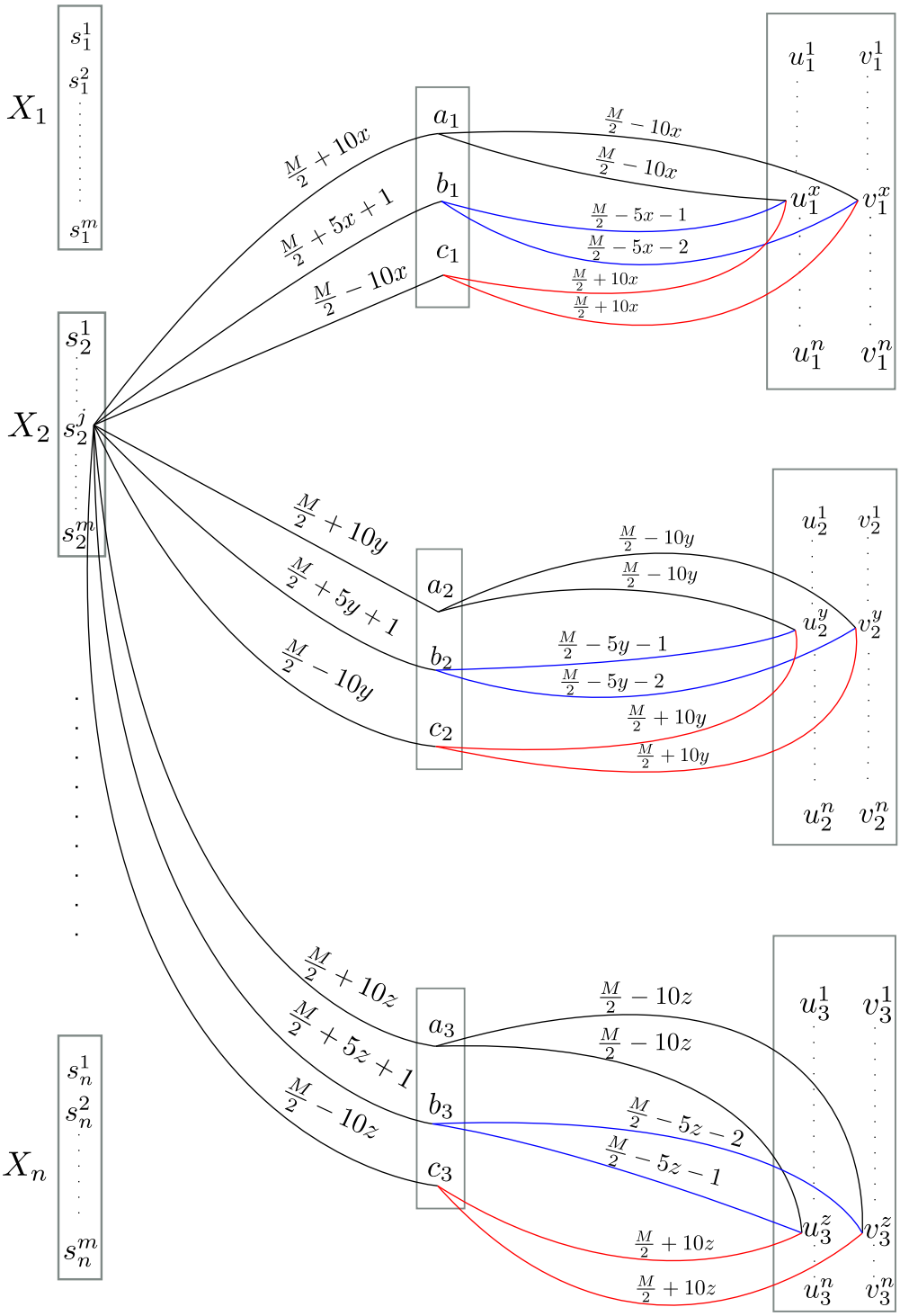


Figure 1 An example of the reduction from 3-Dimensional Matching to n-Multicolored Resolving Set in which $U = \{1, 2, 3\} \times [n]$ and $\mathcal{F} = \{A_1, ..., A_m\}$. Here we only draw the corresponding paths and resolved pairs of the tuple $A_j = \{(1, x), (2, y), (3, z)\}$.

Proof. ($\Leftarrow$) For an arbitrary tuple $A_i = \{(1,x),(2,y),(3,z)\}$, according to Lemma 2, pairs $\{u_1^x, v_1^x\}$,$\{u_2^y, v_2^y\}$ and $\{u_3^z, v_3^z\}$ are all resolved by s_i^j for every $i \in [n]$. Suppose that the given instance of 3-DIMENSIONAL MATCHING is a yes-instance, that is, there exists $A_{j_1}, ..., A_{j_n}$ satisfying that $\bigcup_{h=1}^{n} A_{j_h} = U$. It follows that $S = \{s_h^{j_h} : h \in [n]\}$ is a solution for the constructed instance of n-MULTICOLORED RESOLVING SET.

($\Rightarrow$) Let $S = \{s_h^{j_h} : h \in [n]\}$ be a solution for the constructed instance of n-MULTICOLORED RESOLVING SET. For an arbitrary pair $\{u_r^x, v_r^x\}$, since it is resolved by some $s_{h'}^{j_{h'}} \in S$, according to Lemma 2, $(r,x) \in A_{j_{h'}}$. As a result, $\{A_{j_h} : h \in [n]\}$ is a solution for the instance of 3-DIMENSIONAL MATCHING. ◀

It is well-known that the treewidth of a graph is bounded by the size of a minimum feedback vertex set of the graph. We can easily observe that W is a feedback vertex set of size 9 for G. It follows that the treewidth of G is at most 10. Then we have the following lemma.

▶ **Lemma 4.** *k-Multicolored Resolving Set is NP-hard even on graphs of treewidth at most* 10.

4 Reduction from Multicolored Resolving Set to Metric Dimension

In this section, we create in polynomial time an instance (G', k) of METRIC DIMENSION, which is equivalent to the instance $(G, n, \chi, \mathcal{P})$ of n-MULTICOLORED RESOLVING SET we created in last section. Roughly speaking, the reduction consists in adding gadgets on base of the constructed instance $(G, n, \chi, \mathcal{P})$ to solve the following two issues: (1) the solution for METRIC DIMENSION could contain vertices not in any set of χ or more than one vertex from some set of χ, which would spoil the desired reduction; (2) we did not make sure that every pair of distinct vertices are resolved by the solution in an instance of n-MULTICOLORED RESOLVING SET. We find that similar strategies to those in [2] can be used to solve these two issues. More specifically, we solve the first issue by adding *forced set gadgets*. One such gadget contains two pairs of vertices such that they are only resolved simultaneously by a vertex of X_i (where it is attached). We solve the second issue by adding *forced vertex gadgets*. One such gadget contains a pair of pendant neighboring vertices (*false twins*), both of which are also adjacent to an identical vertex. Such construction forces at least one vertex of the false twins to be chosen in the solution. The chosen vertices (*forced vertices*) are designed to resolve the remaining unresolved vertex pairs. Besides, we need to add a number of extra paths and set appropriate budget k to make sure that the reduction works as described above.

4.1 Construction of the forced set gadgets

Let $(G, n, \chi, \mathcal{P})$ be an instance of n-MULTICOLORED RESOLVING SET that we created in last section. For every $X_i \in \chi$ ($i \in [n]$), we add two pairs of isolated vertices $\{p_i^1, q_i^1\}$ and $\{p_i^2, q_i^2\}$. Then we add two vertices π_i^1 and π_i^2 such that p_i^1, q_i^1 are adjacent to π_i^1, p_i^2, q_i^2 are adjacent to π_i^2. The vertex triples p_i^1, q_i^1, π_i^1 and p_i^2, q_i^2, π_i^2 ($i \in [n]$) form a forced set gadget. Then we create a path $P(s_i^j, p_i^1)$ of length $20(n+1)$ from s_i^j to p_i^1 and create a path $P(s_i^j, p_i^2)$ of length $20(n+1)$ from s_i^j to p_i^2 for each $i \in [n], j \in [m]$. In order to make sure that a vertex can resolve p_i^1, q_i^1 and p_i^2, q_i^2 simultaneously if and only if it belongs to X_i, we need to create 4 paths of length $20(n+1)$ from π_i^1 to $N_{s_i^j}(s_i^j, a_r)$, from π_i^1 to $N_{s_i^j}(s_i^j, b_r)$, from π_i^1 to $N_{s_i^j}(s_i^j, c_r)$ and from π_i^1 to $N_{s_i^j}(s_i^j, p_i^2)$ respectively for each $i \in [n]$, $j \in [m]$ and $r \in \{1,2,3\}$. For simplicity, we name the four paths as $P^1(i,j,a_r)$, $P^1(i,j,b_r)$, $P^1(i,j,c_r)$ and $P^1(i,j,p_i^2)$ respectively.

Symmetrically, we need to create 4 paths of length $20(n+1)$ from π_i^2 to $N_{s_i^j}(s_i^j, a_r)$, from π_i^2 to $N_{s_i^j}(s_i^j, b_r)$, from π_i^2 to $N_{s_i^j}(s_i^j, c_r)$ and from π_i^2 to $N_{s_i^j}(s_i^j, p_i^1)$ respectively for each $i \in [n]$ and $r \in \{1,2,3\}$. For simplicity, we name the four paths as $P^2(i,j,a_r)$, $P^2(i,j,b_r)$, $P^2(i,j,c_r)$ and $P^2(i,j,p_i^1)$ respectively. Let $\Pi^h(i,j,r) = \{P^h(i,j,a_r), P^h(i,j,b_r), P^h(i,j,c_r), P^h(i,j,p_i^{3-h})\}$ for $i \in [n], j \in [m], r \in \{1,2,3\}, h \in \{1,2\}$.

This completes the construction of the first phase.

4.2 Construction of the forced vertex gadgets

A forced vertex gadget consists of a triangle, namely three vertices such that each vertex is adjacent to the other two vertices. Two vertices of the triangle are false twins whose degrees are exactly 2 and we call the other vertex in the triangle the *connecting vertex* of the gadget. When we say that we add a forced vertex gadget F to a vertex v, we mean that we create a forced vertex gadget F such that v is identified with the connecting vertex of F. For each $i \in [n], j \in [m], r \in \{1,2,3\}, h \in \{1,2\}$, we add a forced vertex gadget $F^h(i,j,a_r)$ to $N^h_{\pi_h}(i,j,a_r)$, $F^h(i,j,b_r)$ to $N^h_{\pi_h}(i,j,b_r)$, $F^h(i,j,c_r)$ to $N^h_{\pi_h}(i,j,c_r)$ and $F^h(i,j,p_i^{3-h})$ to $N^h_{\pi_h}(i,j,,p_i^{3-h})$ respectively.

In order to make sure that the false twins of $F^h(i,j,b_r)$ for $i \in [n], j \in [m], r \in \{1,2,3\}, h \in \{1,2\}$ do not resolve any vertex pair of $\mathcal{P}$, we create a path $P(\pi_i^h, a_r)$ and a path $P(\pi_i^h, c_r)$ both of length $10(n+1)$ for $i \in [n]$, $h \in \{1,2\}$ and $r \in \{1,2,3\}$.

For each $i \in [n], j \in [m], r \in \{1,2,3\}, h \in \{1,2\}$, we add a forced vertex gadget $F(\pi_i^h, a_r)$ to $N_{a_r}(\pi_i^h, a_r)$ and a forced vertex gadget $F(\pi_i^h, c_r)$ to $N_{c_r}(\pi_i^h, c_r)$. For each $i \in [n], j \in [m], r \in \{1,2,3\}$, we add a forced vertex gadget $F(s_i^j, a_r)$ to $N_{a_r}(s_i^j, a_r)$ and a forced vertex gadget $F(s_i^j, c_r)$ to $N_{c_r}(s_i^j, c_r)$.

Let $\mathrm{mid}(P^h(i,j,p_i^{3-h}))$ be the middle vertex of $P^h(i,j,p_i^{3-h})$ for $i \in [n], j \in [m], h \in \{1,2\}$. In order to make sure that the false twins of $F^h(i,j,p_i^{3-h})$ do not resolve the vertex pair $\{p_i^{3-h}, q_i^{3-h}\}$, create a path $P(q_i^h, \mathrm{mid}(P^{3-h}(i,j,p_i^h)))$ from q_i^h to $\mathrm{mid}(P^{3-h}(i,j,p_i^h))$ of length $|P^{3-h}(i,j,p_i^h)|/2 + |P(s_i^j, p_i^h)| - 1$. Then add a forced vertex gadget $F^{mid}(i,j,h)$ to $\mathrm{mid}(P^h(i,j,p_i^{3-h}))$.

For $i \in [n], j \in [m], r \in \{1,2,3\}, h \in \{1,2\}$, add a forced vertex gadget $F^{ecc}(i,j,h,r)$ to the vertex $x \in P^h(i,j,a_r)$ such that $\mathrm{dist}(\pi_i^h, x) = 10(n+1)+1$.

For each $i \in [n], r \in \{1,2,3\}$, create two forced vertex gadgets $F^1(u_r^i, v_r^i)$ and $F^2(u_r^i, v_r^i)$ for the vertex pair $\{u_r^i, v_r^i\} \in \mathcal{P}_r$. Then create an edge from the connecting vertex of $F^1(u_r^i, v_r^i)$ to u_r^i, to v_r^i, to $N_{u_r^i}(a_r, u_r^i)$ and to $N_{u_r^i}(c_r, u_r^i)$ respectively for $i \in [n], r \in \{1,2,3\}$. Create an edge from the connecting vertex of $F^2(u_r^i, v_r^i)$ to u_r^i, to v_r^i, to the vertex x such that $x \in P(a_r, u_r^i)$ and $\mathrm{dist}(x, u_r^i) = 2$, and to the vertex y such that $y \in P(c_r, u_r^i)$ and $\mathrm{dist}(y, u_r^i) = 2$. This completes the construction of the second phase.

Finally, let G' be the graph constructed by above two phases and set $k = 34nm + 19n$. This finishes constructing the instance (G', k) of Metric Dimension. Figure 2 shows a part of G'.

4.3 Soundness of the reduction

First, we define the vertex sets to be used in the following parts. Recall that $X_i = \{s_i^1, ..., s_i^m\}$. For every $i \in [n], r \in \{1,2,3\}, h \in \{1,2\}$, let

$$U_i^h = \bigcup_{j \in [m]} P(s_i^j, p_i^h),$$

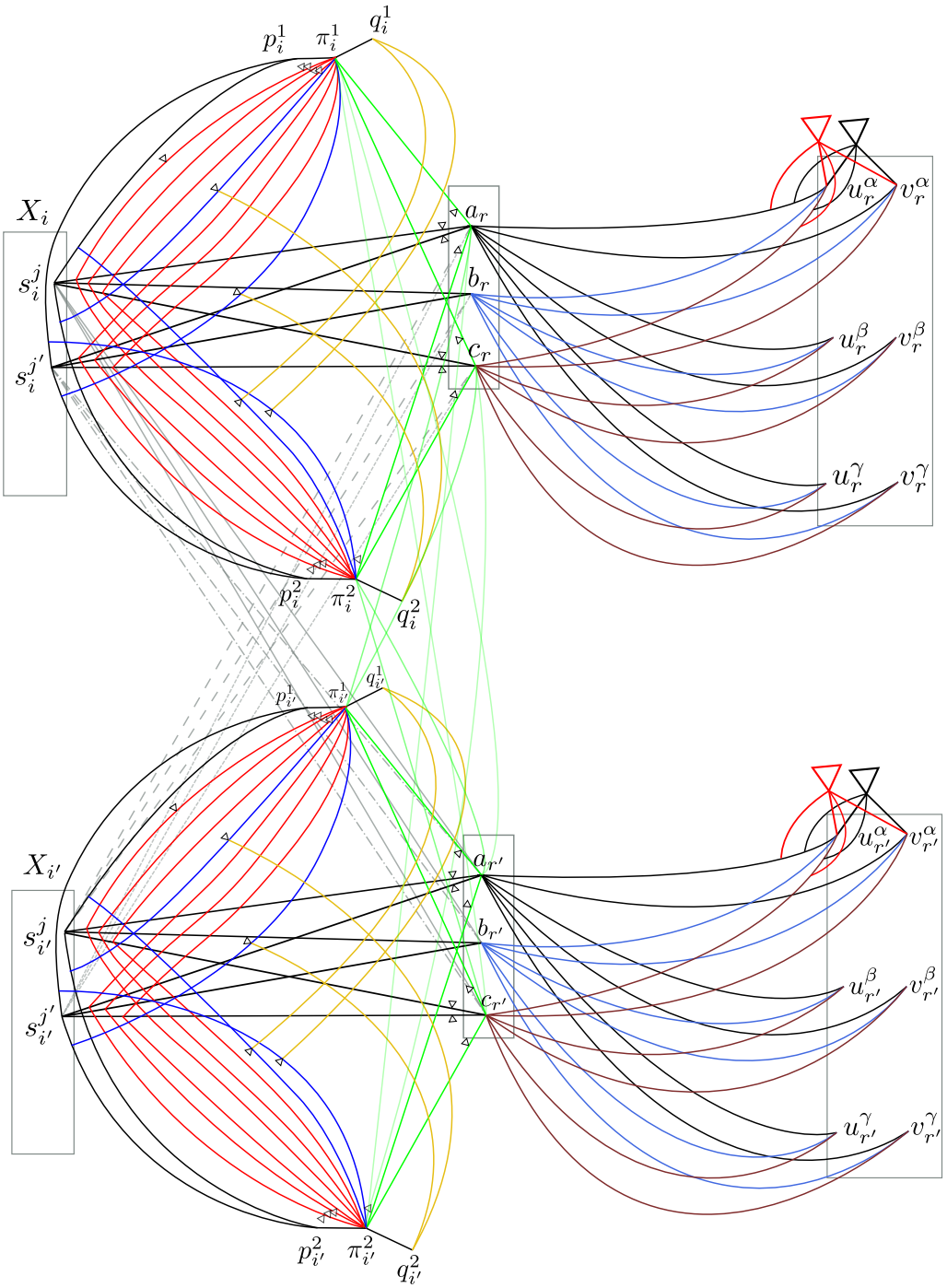


Figure 2 An example showing a part of G'. Triangles represent corresponding forced vertex gadgets. For clarity, some forced vertex gadgets do not appear on the figure. Dotted or dashed lines are used in order for cleanness of the figure.

$$H_{i,r} = \bigcup_{j\in[m]} P(s_i^j, a_r) \cup P(s_i^j, b_r) \cup P(s_i^j, c_r),$$

$$S_i^h = \bigcup_{r\in\{1,2,3\}} P(\pi_i^h, a_r) \cup P(\pi_i^h, c_r),$$

$$L_i^h = \bigcup_{j\in[m]} P(q_i^h, \mathrm{mid}(P^{3-h}(i,j,p_i^h))),$$

$$R_r = \bigcup_{i\in[n]} P(a_r, u_r^i) \cup P(a_r, v_r^i) \cup P(b_r, u_r^i) \cup P(b_r, v_r^i) \cup P(c_r, u_r^i) \cup P(c_r, v_r^i), \text{ and}$$

$$\Pi^h(i,j,r) = P^h(i,j,a_r) \cup P^h(i,j,b_r) \cup P^h(i,j,c_r) \cup P^h(i,j,p_i^{3-h}).$$

For every $i \in [n]$, let

$$U_i = \bigcup_{h\in\{1,2\}} U_i^h \qquad H_i = \bigcup_{r\in\{1,2,3\}} H_{i,r} \qquad S_i = \bigcup_{h\in\{1,2\}} S_i^h$$

$$L_i = \bigcup_{h\in\{1,2\}} L_i^h \qquad \Pi_i = \bigcup_{j\in[m],r\in\{1,2,3\},h\in\{1,2\}} \Pi^h(i,j,r).$$

Let $\mathcal{F}$ be the union of all forced vertex gadgets, i.e.

$$\begin{aligned}\mathcal{F} = \bigcup_{i\in[n],j\in[m],r\in\{1,2,3\},h\in\{1,2\}} & (F(s_i^j, a_r) \cup F(s_i^j, c_r) \cup F(\pi_i^h, a_r) \cup F(\pi_i^h, c_r) \\ & \cup F^h(u_r^i, v_r^i) \cup F^h(i,j,a_r) \cup F^h(i,j,b_r) \cup F^h(i,j,c_r) \\ & \cup F^h(i,j,p_i^{3-h}) \cup F^{mid}(i,j,h) \cup F^{ecc}(i,j,h,r)).\end{aligned}$$

Next we introduce a lemma about forced set gadgets and this lemma is important for the correctness of the reduction.

▶ **Lemma 5.** *The following three statements are true for the instance (G', k).*

(a) *The vertex s_i^j for $i \in [n], j \in [m]$ resolves both pairs $\{p_i^1, q_i^1\}$ and $\{p_i^2, q_i^2\}$. Moreover, s_i^j does not resolve any vertex pair $\{p_{i'}^h, q_{i'}^h\}$ such that $i' \in [n], h \in \{1,2\}$ and $i' \neq i$.*

(b) *The vertices of any forced vertex gadget do not resolve any vertex pair of $\{\{p_i^h, q_i^h\} \mid i \in [n], h \in \{1,2\}\}$.*

(c) *Any vertex $v \in V(G') \setminus (X_i \cup \mathcal{F})$ resolves at most one vertex pair of $\{\{p_i^h, q_i^h\} \mid i \in [n], h \in \{1,2\}\}$.*

Proof. By the construction of G', $\mathrm{dist}(s_i^j, q_i^h) = |P(s_i^j, p_i^h)| + 2 = 20(n+1) + 2 > \mathrm{dist}(s_i^j, p_i^h)$ for $i \in [n], j \in [m]$ and $h \in \{1,2\}$. Thus any vertex of X_i resolves both pairs $\{p_i^1, q_i^1\}$ and $\{p_i^2, q_i^2\}$ for $i \in [n]$. For a vertex pair $\{p_{i'}^{h'}, q_{i'}^{h'}\}$ such that $i' \neq i$, there is a shortest path from s_i^j to $p_{i'}^{h'}$ or $q_{i'}^{h'}$ going through $c_{r'}$ and $\pi_{i'}^{h'}$ with some integer $r' \in \{1,2,3\}$. Thus a vertex $s \in X_i$ resolves exactly two vertex pairs of $\{\{p_i^h, q_i^h\} : i \in [n], h \in \{1,2\}\}$.

First we claim that vertices of $\mathcal{F}$ do not resolve any vertex pair $\{p_{i'}^{h'}, q_{i'}^{h'}\}$ for $i' \in [n], h' \in \{1,2\}$. For any vertex $v \in F^h(u_r^i, v_r^i)$ for $i \in [n], r \in \{1,2,3\}, h \in \{1,2\}$, there is a shortest path from v to $p_{i'}^{h'}$ or $q_{i'}^{h'}$ going through a_r and $\pi_{i'}^{h'}$. Thus v does not resolve any vertex pair $\{p_{i'}^{h'}, q_{i'}^{h'}\}$ for $i' \in [n], h' \in \{1,2\}$. For any vertex $v \in F^{mid}(i,j,h) \cup F^{ecc}(i,j,h,r)$ for $i \in [n], j \in [m], h \in \{1,2\}, r \in \{1,2,3\}$, we can see that $\mathrm{dist}(v, p_{i'}^{h'}) = \mathrm{dist}(v, q_{i'}^{h'})$ with $i' = i$. There is a shortest path from v to $p_{i'}^{h'}$ or $q_{i'}^{h'}$ going through π_i^h, a_r and $\pi_{i'}^{h'}$ with $i' \neq i$. Thus v does not resolve any vertex pair $\{p_{i'}^{h'}, q_{i'}^{h'}\}$ for $i' \in [n], h' \in \{1,2\}$. For any vertex $v \in \mathcal{F} \setminus \bigcup_{i\in[n],j\in[m],r\in\{1,2,3\},h\in\{1,2\}} (F^h(u_r^i, v_r^i) \cup F^{ecc}(i,j,h,r) \cup F^{mid}(i,j,h))$, there

is a shortest path from v to $p_{i'}^{h'}$ or $q_{i'}^{h'}$ going through $\pi_{i'}^{h'}$ with $i' = i$. There is a shortest path from v to $p_{i'}^{h'}$ or $q_{i'}^{h'}$ going through c_r (or a_r) and $\pi_{i'}^{h'}$ with $i' \neq i$. Thus v does not resolve any pair $\{p_{i'}^{h'}, q_{i'}^{h'}\}$. As a result, vertices of $\mathcal{F}$ do not resolve any vertex pair $\{p_{i'}^{h'}, q_{i'}^{h'}\}$ for $i' \in [n], h' \in \{1,2\}$.

Then we show that any vertex $v \in V(G') \setminus (X_i \cup \mathcal{F})$ resolves at most one pair of $\{p_i^1, q_i^1\}$ and $\{p_i^2, q_i^2\}$.

For a vertex $v \in U_i^h \setminus X_i$ for $i \in [n], h \in \{1,2\}$, $\mathrm{dist}(v, p_i^h) = \mathrm{dist}(v, q_i^h) - 2 < \mathrm{dist}(v, q_i^h)$. $\mathrm{dist}(v, q_i^{3-h}) = \mathrm{dist}(v, N_{s_i^j}(s_i^j, p_i^h)) + |P^{3-h}(i, j, p_i^h)| + 1 = \mathrm{dist}(v, p_i^{3-h})$. For a vertex pair $\{p_{i'}^{h'}, q_{i'}^{h'}\}$ such that $i' \neq i$, there is a shortest path from v to $p_{i'}^{h'}$ or $q_{i'}^{h'}$ going through $\pi_{i'}^{h'}$. Thus $v \in U_i^h \setminus X_i$ for $i \in [n], h \in \{1,2\}$ resolves exactly one vertex pair of $\{\{p_i^h, q_i^h\} : i \in [n], h \in \{1,2\}\}$.

Let $P(\mathrm{mid}(P^{3-h}(i,j,p_i^h)), N_{s_i^j}(s_i^j, p_i^h))$ be the subpath of $P^{3-h}(i,j,p_i^h)$ from $\mathrm{mid}(P^{3-h}(i,j,p_i^h))$ to $N_{s_i^j}(s_i^j, p_i^h)$. Let $\Lambda_i^h = (\bigcup_{j\in[m]} P(\mathrm{mid}(P^{3-h}(i,j,p_i^h)), N_{s_i^j}(s_i^j, p_i^h))) \setminus \{\mathrm{mid}(P^{3-h}(i,j,p_i^h)) \mid j \in [m]\}$. For a vertex $v \in \Lambda_i^h$ for $i \in [n], h \in \{1,2\}$, $\mathrm{dist}(v, p_i^h) = \mathrm{dist}(v, q_i^h) - 2 < \mathrm{dist}(v, q_i^h)$. $\mathrm{dist}(v, q_i^{3-h}) = \mathrm{dist}(v, \pi_i^{3-h}) + 1 = \mathrm{dist}(v, p_i^{3-h})$. For a vertex pair $\{p_{i'}^{h'}, q_{i'}^{h'}\}$ such that $i' \neq i$, there is a shortest path from v to $p_{i'}^{h'}$ or $q_{i'}^{h'}$ going through $\pi_{i'}^{h'}$. Thus $v \in \Lambda_i^h$ for $i \in [n], h \in \{1,2\}$ resolves exactly one vertex pair of $\{\{p_i^h, q_i^h\} : i \in [n], h \in \{1,2\}\}$.

For a vertex $v \in L_i^h \setminus \{\mathrm{mid}(P^h(i,j,p_i^{3-h})) \mid j \in [m]\}$ for $i \in [n], h \in \{1,2\}$, $\mathrm{dist}(v, q_i^h) = \mathrm{dist}(v, p_i^h) - 2 < \mathrm{dist}(v, p_i^h)$. There is a shortest path from v to p_i^{3-h} or q_i^{3-h} going through π_i^{3-h}. For a vertex pair $\{p_{i'}^{h'}, q_{i'}^{h'}\}$ such that $i' \neq i$, there is a shortest path from v to $p_{i'}^{h'}$ or $q_{i'}^{h'}$ going through $\pi_{i'}^{h'}$. Thus $v \in L_i^h \setminus \{\mathrm{mid}(P^h(i,j,p_i^{3-h})) \mid j \in [m]\}$ for $i \in [n], h \in \{1,2\}$ resolves exactly one vertex pair of $\{\{p_i^h, q_i^h\} : i \in [n], h \in \{1,2\}\}$.

For a vertex $v \in \Pi_i \cup S_i \cup H_i \setminus (X_i \cup \Lambda_i^1 \cup \Lambda_i^2)$ for $i \in [n]$, there is a shortest path from v to $p_{i'}^{h'}$ or $q_{i'}^{h'}$ going through $\pi_{i'}^{h'}$ with $i = i', h' \in \{1,2\}$. For a vertex pair $\{p_{i'}^{h'}, q_{i'}^{h'}\}$ such that $i' \neq i$, there is a shortest path from v to $p_{i'}^{h'}$ or $q_{i'}^{h'}$ going through $\pi_{i'}^{h'}$. Thus v does not resolve any vertex pair of $\{\{p_i^h, q_i^h\} : i \in [n], h \in \{1,2\}\}$.

For a vertex $v \in R_r$ for $r \in \{1,2,3\}$, there is a shortest path from v to p_i^h or q_i^h for $i \in [n], h \in \{1,2\}$ going through a_r and π_i^h. Thus v does not resolve any vertex pair of $\{\{p_i^h, q_i^h\} : i \in [n], h \in \{1,2\}\}$. This completes the proof for the lemma. ◀

By the properties of false twins, we need to choose exactly one vertex of the false twins (arbitrarily) of every forced vertex gadget in the resolving set of G', which we call a *forced vertex*. For convenience, we use $f(\cdot)$ to represent the chosen forced vertex of the corresponding gadget $F(\cdot)$. Then we have the following lemma.

▶ **Lemma 6.** *The forced vertices do not resolve any vertex pair $\{u_r^i, v_r^i\} \in \mathcal{P}$ for $r \in \{1,2,3\}$ and $i \in [n]$.*

Proof. We fix arbitrary integers $i \in [n], j \in [m], r \in \{1,2,3\}, h \in \{1,2\}$. For the forced vertex $f^h(i,j,a_r)$, $\mathrm{dist}(f^h(i,j,a_r), u_{r'}^{i'}) = 2 + |P(\pi_i^h, a_{r'})| + |P(a_{r'}, u_{r'}^{i'})| = 2 + |P(\pi_i^h, a_{r'})| + |P(a_{r'}, v_{r'}^{i'})| = \mathrm{dist}(f^h(i,j,a_r), v_{r'}^{i'})$ for $i' \in [n], r' \in \{1,2,3\}$. Thus $f^h(i,j,a_r)$ does not resolve any vertex pair of $\mathcal{P}$. Similarly, the forced vertices $f^h(i,j,b_r)$, $f^h(i,j,c_r)$ and $f^h(i,j,p_i^{3-h})$ do not resolve any vertex pair of $\mathcal{P}$. For the forced vertex $f^{mid}(i,j,h)$, $\mathrm{dist}(f^{mid}(i,j,h), u_{r'}^{i'}) = \mathrm{dist}(f^{mid}(i,j,h), v_{r'}^{i'}) = |P^h(i,j,p_i^{3-h})|/2 + |P(\pi_i^h, a_{r'})| + |P(a_{r'}, u_{r'}^{i'})|$. Thus $f^{mid}(i,j,h)$ does not resolve any vertex pair of $\mathcal{P}$. For the forced vertex $f^{ecc}(i,j,h,r)$, $\mathrm{dist}(f^{ecc}(i,j,h,r), u_{r'}^{i'}) = \mathrm{dist}(f^{ecc}(i,j,h,r), v_{r'}^{i'}) = 10(n+1) + 1 + |P(\pi_i^h, a_{r'})| + |P(a_{r'}, u_{r'}^{i'})|$. Thus $f^{ecc}(i,j,h,r)$ does not resolve any vertex pair of $\mathcal{P}$.

We fix arbitrary integers $i \in [n], j \in [m], r \in \{1,2,3\}$. For the forced vertex $f(s_i^j, a_r)$, $\mathrm{dist}(f(s_i^j, a_r), u_r^{i'}) = 2 + |P(a_r, u_r^{i'})| = 2 + |P(a_r, v_r^{i'})| = \mathrm{dist}(f(s_i^j, a_r), v_r^{i'})$ for $i' \in [n]$. For the forced vertex $f(s_i^j, c_r)$, $\mathrm{dist}(f(s_i^j, c_r), u_r^{i'}) = 2 + |P(c_r, u_r^{i'})| = 2 + |P(c_r, v_r^{i'})| =$

$\text{dist}(f(s_i^j, c_r), v_r^{i'})$ for $i' \in [n]$. Thus $f(s_i^j, a_r)$ and $f(s_i^j, c_r)$ do not resolve any vertex pair of $\mathcal{P}_r$. Similarly, $f(\pi_i^h, a_r)$ and $f(\pi_i^h, c_r)$ for $i \in [n], h \in \{1,2\}, r \in \{1,2,3\}$ do not resolve any vertex pair of $\mathcal{P}_r$. For vertex pairs of $\mathcal{P}_{r'}$ with $r' \in \{1,2,3\}$ and $r' \neq r$, $\text{dist}(f(s_i^j, a_r), u_{r'}^{i'}) = 2+|P(a_r, \pi_i^1)|+|P(a_{r'}, \pi_i^1)|+|P(a_{r'}, u_r^{i'})| = 2+|P(a_r, \pi_i^1)|+|P(a_{r'}, \pi_i^1)|+|P(a_{r'}, v_r^{i'})| = \text{dist}(f(s_i^j, a_r), v_{r'}^{i'})$ for $i' \in [n]$. $\text{dist}(f(s_i^j, c_r), u_{r'}^{i'}) = 2 + |P(c_r, \pi_i^1)| + |P(a_{r'}, \pi_i^1)| + |P(a_{r'}, u_r^{i'})| = 2 + |P(c_r, \pi_i^1)| + |P(a_{r'}, \pi_i^1)| + |P(a_{r'}, v_r^{i'})| = \text{dist}(f(s_i^j, a_r), v_{r'}^{i'})$ for $i' \in [n]$. Thus $f(s_i^j, a_r)$ and $f(s_i^j, c_r)$ do not resolve any vertex pair of $\mathcal{P}_{r'}$.

We fix arbitrary integers $i \in [n], r \in \{1,2,3\}$. For the forced vertex $f^1(u_r^i, v_r^i)$ or $f^2(u_r^i, v_r^i)$, obviously it does not resolve the vertex pair $\{u_r^i, v_r^i\}$. For a vertex pair $\{u_r^{i'}, v_r^{i'}\}$ with $i' \in [n]$ and $i' \neq i$, $\text{dist}(f^1(u_r^i, v_r^i), u_r^{i'}) = 2 + |P(a_r, u_r^i)| - 1 + |P(a_r, u_r^{i'})| = 2 + |P(a_r, u_r^i)| - 1 + |P(a_r, v_r^{i'})| = \text{dist}(f^1(u_r^i, v_r^i), v_r^{i'})$. For a vertex pair $\{u_{r'}^{i'}, v_{r'}^{i'}\}$ with $i' \in [n]$ and $r' \in \{1,2,3\}$ and $r' \neq r$, $\text{dist}(f^1(u_r^i, v_r^i), u_{r'}^{i'}) = 2 + |P(a_r, u_r^i)| - 1 + |P(\pi_i^1, a_r)| + |P(\pi_i^1, a_{r'})| + |P(a_{r'}, u_{r'}^{i'})| = \text{dist}(f^1(u_r^i, v_r^i), v_{r'}^{i'})$. As a result, $f^1(u_r^i, v_r^i)$ does not resolve any vertex pair of $\mathcal{P}$. For a vertex pair $\{u_r^{i'}, v_r^{i'}\}$ with $i' \in [n]$ and $i' \neq i$, $\text{dist}(f^2(u_r^i, v_r^i), u_r^{i'}) = 2 + |P(a_r, u_r^i)| - 2 + |P(a_r, u_r^{i'})| = 2 + |P(a_r, u_r^i)| - 2 + |P(a_r, v_r^{i'})| = \text{dist}(f^2(u_r^i, v_r^i), v_r^{i'})$. For a vertex pair $\{u_{r'}^{i'}, v_{r'}^{i'}\}$ with $i' \in [n]$, $r' \in \{1,2,3\}$ and $r' \neq r$, $\text{dist}(f^2(u_r^i, v_r^i), u_{r'}^{i'}) = 2+|P(a_r, u_r^i)|-2+|P(\pi_i^1, a_r)|+|P(\pi_i^1, a_{r'})|+|P(a_{r'}, u_{r'}^{i'})| = \text{dist}(f^2(u_r^i, v_r^i), v_{r'}^{i'})$. As a result, $f^2(u_r^i, v_r^i)$ does not resolve any vertex pair of $\mathcal{P}$. This completes the proof for the lemma. ◀

▶ **Lemma 7** (Soundness). *If G' has a resolving set of size at most $34nm + 19n$, then $(G, n, \chi, \mathcal{P})$ is a yes-instance.*

Proof. Suppose that $\mathcal{S}$ is a resolving set for G' of size at most $34nm + 19n$. Let $\hat{\mathcal{S}} = \mathcal{S} \cap X$. (Recall that $X = \bigcup_{i=1}^{n} \{s_i^1, ..., s_i^m\}$.) We claim that $\hat{\mathcal{S}}$ is solution for $(G, n, \chi, \mathcal{P})$. Note that for the false twins $\{u, u'\}$ of a forced vertex gadget, no vertex resolves the vertex pair $\{u, u'\}$ except u (or u'). It follows that $\mathcal{S}$ contains $34nm + 18n$ forced vertices since there are $34nm+18n$ forced vertex gadgets in G'. Since X has no intersection with the vertex set of all forced vertex gadgets, $|\hat{\mathcal{S}}| \leq n$. By Lemma 5, we get that $|\hat{\mathcal{S}} \cap X_i| = 1$ for each $i \in [n]$. Thus $|\hat{\mathcal{S}}| = n$. By Lemma 6 and the assumption that $\mathcal{S}$ is a resolving set for G', $\hat{\mathcal{S}}$ resolves every pair $\{u_r^i, v_r^i\}$ in G' for $r \in \{1,2,3\}$ and $i \in [n]$. We can check that the distance between s_i^j and $u_r^{i'}$ in G' (and the distance between s_i^j and $v_r^{i'}$ in G') for $i \in [n], j \in [m], i' \in [n], r \in \{1,2,3\}$ is the same as that in G. Thus $\hat{\mathcal{S}}$ is a solution for $(G, n, \chi, \mathcal{P})$. ◀

4.4 Treewidth bound of the graph

Since the completeness proof takes a large amount of space, before proceeding to that, we first show that G' is of constant treewidth. In fact, we will prove a slightly stronger statement that G' is of constant pathwidth by giving a search strategy with a constant number of searchers.

▶ **Lemma 8.** *The pathwidth of G' is at most 24.*

Proof. Following the characterization of pathwidth by Kirousis and Papadimitriou [11], we give a search strategy with 25 searchers. First, we put 9 searchers on $\bigcup_{r\in\{1,2,3\}}\{a_r, b_r, c_r\}$. The 9 searchers remain there until the end of the whole searching process. The searching process consists of two phases. We search the "left" part of G' in the first phase and the "right" part of G' in the second phase.

The first phase of the searching process consists of n rounds. At the beginning of the i-th round ($i \in [n]$), we put 6 searchers on $\bigcup_{h\in\{1,2\}}\{p_i^h, q_i^h, \pi_i^h\}$. Here when we say that we clean a path, this means that there are already two searchers guarding at the endpoints (or the

neighbor of the endpoints) of this path and we use 3 extra searchers x, y, z such that x, y move alternately from one end of the path to the other end to clean the edges of the path. When a searcher, say x arrives at the connecting point of a forced vertex gadget, we put y, z on the false twins of this forced vertex gadget to clean the edges of this gadget and then after removing y, z, put y ahead of x to continue the alternating process unless x reaches the endpoint of this path. Then for each $j \in [m]$, we

- put 5 vertices on $N_{G'}(s_i^j)$.
- put 2 vertices on $\text{mid}(P^h(i, j, p_i^{3-h}))$ for $h \in \{1, 2\}$.
- use 3 extra searchers to clean the paths $P(s_i^j, p_i^h)$ for $h \in \{1, 2\}$, the paths $P(s_i^j, a_r)$, $P(s_i^j, b_r)$, $P(s_i^j, c_r)$ for $r \in \{1, 2, 3\}$, the paths $P^h(i, j, a_r)$, $P^h(i, j, b_r)$, $P^h(i, j, c_r)$, $P^h(i, j, p_i^{3-h})$ for $h \in \{1, 2\}, r \in \{1, 2, 3\}$, the paths $P(\pi_i^h, a_r)$, $P(\pi_i^h, c_r)$ for $h \in \{1, 2\}, r \in \{1, 2, 3\}$ and the path $P(q_i^h, \text{mid}(P^{3-h}(i, j, p_i^h)))$ for $h \in \{1, 2\}$ successively (including all forced vertex gadgets attached to the vertices on these paths).
- remove the above 10 searchers that are still on the graph.

At the end of the i-th round, we remove the 6 searchers on $\bigcup_{h\in\{1,2\}} \{p_i^h, q_i^h, \pi_i^h\}$.

The second phase of the searching process consists of 3 rounds. During the r-th round ($r \in \{1, 2, 3\}$), we operate as follows. For each $i \in [n]$, we

- put 4 searchers on u_r^i, v_r^i and the connecting point of $F^h(u_r^i, v_r^i)$ for $h \in \{1, 2\}$.
- use 2 extra searchers to clean the paths $P(a_r, u_r^i), P(b_r, u_r^i), P(c_r, u_r^i), P(a_r, v_r^i), P(b_r, v_r^i)$ and $P(c_r, v_r^i)$ (including the forced vertex gadgets $F^h(u_r^i, v_r^i)$ for $h \in \{1, 2\}$ and the incident edges of the connecting vertex of $F^h(u_r^i, v_r^i)$).
- remove the above 6 searchers that are still on the graph.

This completes the description of the the search strategy.

As a result, the node search number of G' is at most 25. It follows that the pathwidth of G' is bounded by 24. ◀

4.5 Completeness of the reduction

For every forced vertex gadget of G', we choose a vertex from the false twins arbitrarily as a forced vertex and let the set of all chosen forced vertices be F. In this section, we show that if $(G, n, \chi, \mathcal{P})$ has a solution $\mathcal{S}$, then $\mathcal{S}' = \mathcal{S} \cup F$ is a resolving set of size at most $34nm + 19n$ for G'. Formally, we will prove the following lemma.

▶ **Lemma 9** (Completeness). *If $(G, n, \chi, \mathcal{P})$ is a yes-instance, then G' has a resolving set of size at most $34nm + 19n$.*

The proof of Lemma 9 consists of a list of lemmas. Suppose that $V(G') = V_1 \cup V_2 \cup ... \cup V_t$. Our general method is to show that for each $i \in [t]$, every internal vertex pair of V_i is resolved by $\mathcal{S}'$ and every vertex pair of $V_{i'} \times V_i$ for each $i' < i$ is resolved by $\mathcal{S}'$. Note that when we mention the vertex pairs of $V_{i'} \times V_i$, we ignore the vertex pairs with two identical vertices by default as it's meaningless in our problem. Due to the page constraint, the proof of Lemma 9 is deferred to the full version of the paper.

5 Conclusion

In this paper, we show that METRIC DIMENSION is NP-hard on graphs of treewidth at most 24. One of the key points in bounding the treewidth of G' is to maintain a vertex separation of constant size. In the first step of our construction, we need 9 vertices to be the vertex separation and convey the choice of the vertices in each color class X_i ($i \in [n]$). It seems hard to show NP-hardness of this problem on graphs of treewidth bounded by a constant

$c \leq 9$ using the techniques in this paper, so we mention this open problem again: is METRIC DIMENSION polynomial-time solvable on graphs of treewidth 2 or series-parallel graphs [1]? Another direction is about the parameterized complexity of METRIC DIMENSION. We ask the following two questions. Is METRIC DIMENSION FPT parameterized by the size of the resolving set on constant treewidth graph? Is METRIC DIMENSION FPT parameterized by both the size of the resolving set and the treewidth of the input graph?

References

1 Rémy Belmonte, Fedor V. Fomin, Petr A. Golovach, and M. S. Ramanujan. Metric dimension of bounded tree-length graphs. *SIAM J. Discret. Math.*, 31(2):1217–1243, 2017. doi:10.1137/16M1057383.

2 Édouard Bonnet and Nidhi Purohit. Metric dimension parameterized by treewidth. In Bart M. P. Jansen and Jan Arne Telle, editors, *14th International Symposium on Parameterized and Exact Computation, IPEC 2019, September 11-13, 2019, Munich, Germany*, volume 148 of *LIPIcs*, pages 5:1–5:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.IPEC.2019.5.

3 Josep Díaz, Olli Pottonen, Maria J. Serna, and Erik Jan van Leeuwen. Complexity of metric dimension on planar graphs. *J. Comput. Syst. Sci.*, 83(1):132–158, 2017. doi:10.1016/j.jcss.2016.06.006.

4 David Eppstein. Metric dimension parameterized by max leaf number. *J. Graph Algorithms Appl.*, 19(1):313–323, 2015. doi:10.7155/jgaa.00360.

5 Leah Epstein, Asaf Levin, and Gerhard J. Woeginger. The (weighted) metric dimension of graphs: Hard and easy cases. *Algorithmica*, 72(4):1130–1171, 2015. doi:10.1007/s00453-014-9896-2.

6 Henning Fernau, Pinar Heggernes, Pim van 't Hof, Daniel Meister, and Reza Saei. Computing the metric dimension for chain graphs. *Inf. Process. Lett.*, 115(9):671–676, 2015. doi:10.1016/j.ipl.2015.04.006.

7 Frank Harary and Robert A Melter. On the metric dimension of a graph. *Ars Combin*, 2(1):191–195, 1976.

8 Sepp Hartung and André Nichterlein. On the parameterized and approximation hardness of metric dimension. In *Proceedings of the 28th Conference on Computational Complexity, CCC 2013, K.lo Alto, California, USA, 5-7 June, 2013*, pages 266–276. IEEE Computer Society, 2013. doi:10.1109/CCC.2013.36.

9 Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

10 Samir Khuller, Balaji Raghavachari, and Azriel Rosenfeld. Landmarks in graphs. *Discret. Appl. Math.*, 70(3):217–229, 1996. doi:10.1016/0166-218X(95)00106-2.

11 Lefteris M Kirousis and Christos H Papadimitriou. Interval graphs and seatching. *Discrete Mathematics*, 55(2):181–184, 1985.

12 Peter J Slater. Leaves of trees. *Congr. Numer*, 14(37):549–559, 1975.

Classification of OBDD Size for Monotone 2-CNFs

Igor Razgon ✉
Department of Computer Science and Information Systems, Birkbeck University of London, UK

Abstract

We introduce a new graph parameter called linear upper maximum induced matching width LU-MIM WIDTH, denoted for a graph G by $lu(G)$. We prove that the smallest size of the OBDD for φ, the monotone 2-CNF corresponding to G, is sandwiched between $2^{lu(G)}$ and $n^{O(lu(G))}$. The upper bound is based on a combinatorial statement that might be of an independent interest. We show that the bounds in terms of this parameter are best possible.

The new parameter is closely related to two existing parameters: linear maximum induced matching width (LMIM WIDTH) and linear special induced matching width (LSIM WIDTH). We prove that LU-MIM WIDTH lies strictly in between these two parameters, being dominated by LSIM WIDTH and dominating LMIM WIDTH. We conclude that neither of the two existing parameters can be used instead of LU-MIM WIDTH to characterize the size of OBDDs for monotone 2-CNFs and this justifies introduction of the new parameter.

2012 ACM Subject Classification Mathematics of computing → Graph theory; Theory of computation → Circuit complexity

Keywords and phrases Ordered Binary Decision Diagrams, Monotone 2-CNFs, Width parameters of graphs, upper and lower bounds

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.25

Acknowledgements I would like to thank the anonymous reviewers for their detailed and insightful comments that helped me to significantly improve the quality of the paper.

1 Introduction

Statement of the results. Monotone 2-CNFs are CNFs with two positive literals per clause. They can be viewed as graphs without isolated vertices. In particular, for such a graph G, $\varphi = \varphi(G)$ is a CNF consisting of clauses $(u \vee v)$ for each $\{u, v\} \in E(G)$. We refer to G as the *underlying graph* of φ.

In this paper we introduce a new graph parameter called (Linear Upper Maximum Induced Matching Width) (LU-MIM WIDTH). This parameter is located 'in-between' of two existing parameters: Linear Maximum Induced Matching Width (LMIM WIDTH) [13] and Linear Special Induced Matching Width (LSIM WIDTH) [8]. We prove that LU-MIM WIDTH captures the size of Ordered Binary Decision Diagrams (OBDDs) for monotone 2-CNFs with a quasipolynomial gap. In particular, we show that $2^{lu(G)} \leq obdd(\varphi) \leq n^{O(lu(G))}$ where $obdd(\varphi)$ is the smallest number of nodes of an OBDD for a monotone 2-CNF φ and $lu(G)$ is the LU-MIM WIDTH of the underlying graph G of φ. The upper bound is based on a combinatorial statement that may be of independent interest. In particular, we exhibit a connection of this statement to the Sauer-Shelah lemma (e.g. Theorem 10.1 in [7]).

We show that the bounds are best possible by demonstrating classes of graphs G_1 and G_2 such that $obdd(\varphi(G_1)) \geq n^{\Omega(lu(G_1))}$ and $obdd(\varphi(G_2)) \leq 2^{O(lu(G_2))}$.

Finally, we prove that LU-MIM WIDTH is located strictly in between LSIM WIDTH and LMIM WIDTH. In particular, we demonstrate classes of graphs G_1 and G_2 such that

- LSIM WIDTH of G_1 is at most 3 while LU-MIM WIDTH is at least $\Omega(n^{1/3})$ and
- LU-MIM WIDTH of G_2 is at most 4 while LMIM WIDTH of G_2 is at least $\Omega(n^{1/2})$.

We conclude from the above dependencies that LSIM WIDTH cannot capture the upper bound of OBDDs for monotone 2-CNFs while LMIM WIDTH cannot capture the lower bound.

Motivation. Monotone CNFs are essentially hypergraphs while monotone 2-CNFs are essentially graphs. Therefore, it is natural to try to characterize the size of models of the corresponding Boolean functions by graph parameters. It is particularly neat if such a parameter can 'capture' the size of a model on a class of CNFs, that is to tightly characterize both upper and lower bounds. It is also desirable for the parameter to be well known as, in this case, existing techniques can be harnessed for determining the value of the parameter for the given class of graphs.

An example of such a neat capturing is characterization of the size of non-deterministic read-once branching programs (1-NBPs) representing monotone 2-CNFs $\varphi(G)$ where G has a bounded degree. In this case, considering the degree constant, the size of the smallest 1-NBP representing $\varphi(G)$ is $2^{\Theta(pw(G))}$ where $pw(G)$ is the pathwidth of G: the upper bound has been established in [3], the lower bound in terms of linear maximum matching width in [10] and it has been shown in [11] that the maximum matching width and pathwidth are linearly related. Since the upper bound of [3] holds for OBDDS, a special case of 1-NBPs, the capturing also works for OBDDs It is natural to ask whether such a capturing is possible for graphs of unbounded degree.

In this paper we address the above question *partially*. First, we obtain the result for OBDDs, a special case of 1-NBPs. Generalization to 1-NBPs is left as an open question. It is important to remark that although, for bounded degrees, the pathwidth captures the sizes of both models, for the case of unbounded degree another parameter might be needed for 1-NBPs. Second, there is a quasipolynomial gap between the upper and lower bounds. We believe that this is still reasonable because the value of the parameter provides a good indication of the size of the resulting OBDD. Besides, we show that for the considered parameter, no tighter capturing is possible. Third, we introduced a new parameter rather than using an existing one. However, this parameter is closely related to existing ones and, as mentioned above, we demonstrate that the related existing parameters cannot be used for the stated purpose.

An additional motivation of the proposed results is that they contribute to understanding the combinatorics of MIM WIDTH, a parameter becoming increasingly popular among graph algorithms researchers (see the related work part for the relevant references).

Related work. Here we overview related results that have not been mentioned in the earlier parts of the introduction.

The size of Decomposable Negation Normal Forms (DDNFs) for monotone 2-CNFs of bounded degree is captured by treewidth. In particular an FPT upper bound for CNFs of bounded (primal) treewidth is proved in Theorem 16 of [2]. A matching lower bound for CNFs of bounded arity and bounded number of variable occurrences follows from the combination of Theorem 8.3 and Lemma 8.4. [1] [1]

A lower bound for OBDDs for monotone CNFs is established in [1]. For 2-CNFs, the lower bound is $2^{\Omega(pw(G)/d^2)}$ where $pw(G)$ and d are the pathwidth and the max-degree of G. The lower bound provided in this paper is better because $lu(G) = \Omega(pw(G)/d)$ due to pathwidth and linear maximum matching width being linearly related [11]. The proof of the $n^{O(lu(G))}$ upper bound is similar in spirit to Lemma 1 of [12].

The MIM-WIDTH [13] has proven useful for design of efficient algorithms for intractable problems for restricted classes of graphs, see for example the recent series of papers [5], [6],[4]. Lower bounds of MIM-WIDTH for several graph classes have been established in [9].

[1] I would like to thank Florent Capelli for pointing the result out to me.

Structure of the paper. Section 2 introduces the necessary background. Section 3 introduces the LU-MIM WIDTH parameter. Section 4 proves upper and lower bounds on the OBDD size. Section 5 proves that, in terms of the parameter, the bounds are essentially tight. Section 6 justifies the introduction of a new parameter by showing that neither LMIM-WIDTH nor LSIM can be used for the purpose of capturing the OBDD size for monotone 2-CNFs. Finally, Section 7 outlines directions of further research.

2 Preliminaries

A *literal* is a Boolean variable or its negation. Throughout this paper, when we refer to a set S of literals, we mean that S is a *proper* set of literals, that is a variable cannot occur in S along with its negation. The set of variables occurring in S is denoted by $Var(S)$. A variable $x \in Var(S)$ can occur in S either *positively*, if $x \in S$ or *negatively*, if $\neg x \in S$. We can also call S an assignment (to $Var(S)$ if the clarification is needed).

We view a *Conjunctive Normal Form* (CNF) as a set of *clauses* and each clause is just a proper set of literals. An assignment S *satisfies* a clause C if $S \cap C \neq \emptyset$. An assignment *satisfies* a CNF φ if $S \cap C \neq \emptyset$ for each $C \in \varphi$. For an assignment S, the CNF $\varphi|_S$ is obtained from φ by removal of all the clauses satisfied by S and removal the occurrences of $Var(S)$ from each remaining clause. We denote by $Var(\varphi)$ the set of all variables occurring in the clauses of φ. Customarily, $|Var(\varphi)|$ is denoted by n.

For a CNF φ, $U \subseteq Var(\varphi)$, let $\mathbf{A}(U) = \mathbf{A}_\varphi(U)$ be the set of all assignments S to U that can be extended to a satisfying assignment of φ. We denote by $\mathbf{BF}(U) = \mathbf{BF}_\varphi(U)$ the set of all Boolean functions represented by $\varphi|_A$ for $A \in \mathbf{A}(U)$.

▶ **Example 1.** Let $\varphi = (x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_2 \vee x_4)$. Let $U = \{x_1, x_2\}$. Then $\mathbf{A}(U) = \{\{x_1, x_2\}, \{x_1, \neg x_2\}, \{\neg x_1, x_2\}\}$. Note that $\{\neg x_1, \neg x_2\}$ is not included in $\mathbf{A}(U)$ because the assignment falsifies the clause $(x_1 \vee x_2)$ and hence cannot be extended to a satisfying assignment of φ. Then $\mathbf{BF}(U)$ is the set of functions on x_3, x_4 represented by the following set of CNFs $\{(x_3 \vee x_4), (x_3), (x_4)\}$.

An *Ordered Binary Decision Diagram* (OBDD) is a popular model for representation of Boolean functions. For the purpose of this paper, we do not need a formal definition of OBBDs because the only fact about OBDDs we use is Proposition 3 but we provide a definition for the sake of completeness.

▶ **Definition 2.** *An OBDD Z is a directed acyclic graph (DAG) with* one source *and* two sinks. *Each non-sink vertex has exactly two outgoing neighbours. The vertices and edges of Z are labelled in the way specified below.*

Each non-sink vertex is labelled with a variable, one of the sinks is labelled with true, the other is labelled with false. Let u be a non-sink node of Z labelled with a variable x. Then one outgoing edge of u is labelled with the positive literal of x, that is x, the other is labelled with the negative literal of x, that is $\neg x$.

The labelling of non-sink nodes also needs to observe two principles: being read-once *and being* ordered. *The read-once property means that in any directed path P of Z the labels of all the non-sink nodes of P are distinct (no variable occurs twice). Being ordered means that there is a permutation $\pi(Z)$ of the variables labelling the nodes of Z so that for any path P from a non-sink node u to a non-sink node v the label of u precedes in $\pi(Z)$ the label of v.*

For a directed path P of Z, we denote by $A(P)$ the set of literals labelling the edges of P. Let $x_1, \ldots, x_n$ be the variables labelling the nodes of Z. The function f_Z represented by Z is defined as follows. Let S be a set of literals with $Var(S) = \{x_1, \ldots, x_n\}$. Then f_Z is true on S if and only if Z has a path P from the source to the true sink such that $A(P) \subseteq S$.

We refer the reader to [14] for an extensive study of OBDDs. For the results of this paper, we only need bounds on $obdd(\varphi)$, the smallest OBDD size (the number of vertices) for a CNF φ as in the next statement that follows from Theorem 3.1.4 of [14].

▶ **Proposition 3.**
1. *Suppose that for each permutation π of $Var(\varphi)$ there is a prefix π' of π* [2] *such that $|\mathbf{BF}(\pi')| \geq m$.* [3] *Then $obdd(\varphi) \geq m$.*
2. *Assume that there is a permutation π of $Var(\varphi)$ such that for every prefix π' of π, $|\mathbf{BF}(\pi')| \leq m$. Then $obdd(\varphi) = O(n * m)$.*

In case of OBDDs representing monotone 2-CNFs upper and lower bounds can be stated in graph theoretical terms as described below. We follow a standard graph-theoretical notation. In particular $G[U]$ denotes the subgraph induced by $U \subseteq V(G)$. $N(U)$ is the set of all neighbours of vertices of U excluding U, the considered graph may be added as a subscript if not clear from the context. The CNF $\{(u \vee v)|\{u, v\} \in E(G)\}$ is denoted by $\varphi(G)$.

▶ **Definition 4.** *Let $U \subseteq V(G)$. We denote by $\mathbf{ISET}(U)$ the family of all the independent subsets of U. Let $V = V(G) \setminus U$. We define $\mathbf{TRACES}(U) = \{N(S) \cap V | S \in \mathbf{ISET}(U)\}$. The subscript G can be used for $\mathbf{TRACES}(U)$ and $\mathbf{ISET}(U)$ if the graph in question is not clear from the context.*

▶ **Example 5.** Let G be a graph with vertices x_1, x_2, x_3, x_4 and edges $\{x_1, x_2\}, \{x_1, x_3\}, \{x_2, x_4\}, \{x_3, x_4\}$. (This is the graph corresponding to the CNF considered in Example 1.) Let $U = \{x_1, x_2\}$. Then $\mathbf{ISET}(U) = \{\emptyset, \{x_1\}, \{x_2\}\}$, $\mathbf{TRACES}(U) = \{\emptyset, \{x_3\}, \{x_4\}\}$.

Combination of Examples 1 and 5 illustrates that $\mathbf{TRACES}_G(U)$ and $\mathbf{BF}_\varphi(U)$ are of the same size where $\varphi = \varphi(G)$. The following lemma shows that this is not a coincidence.

▶ **Lemma 6.** *Let $\varphi = \varphi(G)$. Then $|\mathbf{BF}(U)| = |\mathbf{TRACES}(U)|$.*

Proof. It is not hard to see that $\mathbf{A}(U) = \{A(S)|S \in \mathbf{ISET}(U)\}$ where $A(S)$ is an assignment on U where all the elements of S occur negatively and the rest occur positively. Furthermore, it is not hard to see that $\varphi|_{A(S)}$ is a CNF of the form $\{(u)|u \in N(S) \cap V\} \cup \{(u \vee v)|\{u, v\} \in E(G[V])\}$ (which is equivalent to $\{(u)|u \in N(S) \cap V\} \cup \{(u \vee v)|\{u, v\} \in E(G[V \setminus N(S)])\}$). It follows that for $S_1, S_2 \in \mathbf{ISET}(U)$ and $N(S_1) \cap V = N(S_2) \cap V$, $\varphi|_{A(S_1)} = \varphi|_{A(S_2)}$. Conversely, we need to show that if $N(S_1) \cap V$ and $N(S_2) \cap V$ are distinct then so are the functions of $\varphi|_{A(S_1)}$ and $\varphi|_{A(S_2)}$. Assume w..l.o.g. the existence of $v \in (N(S_1) \cap V) \setminus (N(S_2) \cap V)$ This means that v occurs positively in all satisfying assignments of $\varphi_{A(S_1)}$ but can occur negatively in $\varphi_{A(S_2)}$: just assign positively the rest of the variables. ◀

Finally, we need one more definition.

▶ **Definition 7.** *Let $U, V \subseteq V(G)$. A (U, V)-matching is a matching of G consisting of edges with one end in U and the other in V. Let M be such a matching. We denote by $U(M)$ the set of ends of the edges of M that belong to U. Let S be an independent subset of U. We say that S* enables *an* induced (U, V) *matching if there is an induced (U, V)-matching M with $U(M) = S$.*

---

[2] We think of the permutation as a linear order of $Var(\varphi)$ so a prefix is naturally defined.

[3] Here and in several other places we slightly abuse the notation by using a sequence as a set. The correct use will always be clear from the context.

3 Linear upper induced matching width

In this section we introduce the parameter of *Linear Upper Maximum Induced Matching Width* (LU-MIM WIDTH). In order to present the parameter in the right context we compare it with two existing parameters: Linear Maximum Induced Matching Width (LMIM WIDTH) and Linear Special Induced Matching Width (LSIM WIDTH).

The definition of all three parameters follows the same pattern. First, we fix a permutation $\pi = (v_1, \ldots, v_n)$, denote each $\{v_1, \ldots, v_i\}$ by V_i and define the width of the prefix $(v_1, \ldots, v_i)$ as the largest size of an induced $(V_i, V(G) \setminus V_i)$-matching of some subgraph of G. The difference between the above three parameters is in the choice of the subgraph. The rest of the definition is identical for all the three parameters and also pretty standard: the width of π is the largest width among all the prefixes of π and the width of G is the smallest width among all the permutations.

To define the width of a permutation prefix for LU-MIM WIDTH, we need the notion of an *upper subgraph* introduced in the definition below.

▶ **Definition 8.** *Let $U \subseteq V(G)$ and $V = V(G) \setminus U$. The* upper subgraph G^U *of G w.r.t. U is a spanning subgraph of G with $E(G^U) = E(G) \setminus E(G[V])$.*

In words, G^U is obtained from G by removal of all the edges whose both ends are outside of U. See Figure 1 for an illustration of this notion.

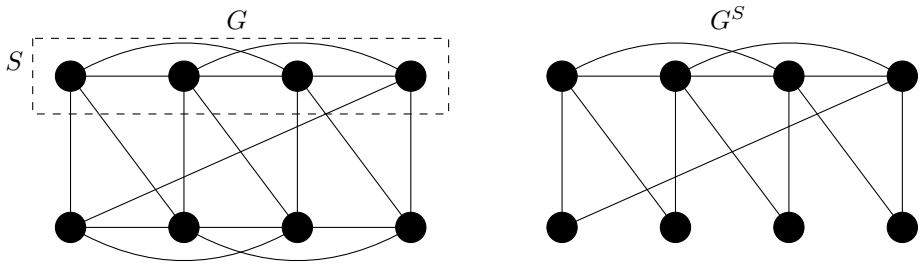


Figure 1 An example of an upper subgraph.

▶ **Definition 9** (LU-MIM WIDTH)**.** *Let $\pi = (v_1, \ldots, v_n)$ be a permutation of $V(G)$ and denote $\{v_1, \ldots, v_i\}$ by V_i. Let r_i be the size of the largest induced $(V_i, V(G) \setminus V_i)$-matching of G^{V_i}. Let $r(\pi) = max_{i=1}^{n} r_i$. The* Linear Upper Induced Matching Width *(LU-MIM WIDTH) of G denoted by $lu(G)$ is the smallest $r(\pi)$ over all the permutations π of $V(G)$. We call a permutation π such that $r(\pi) = lu(G)$ a* witnessing permutation *for $lu(G)$.*

▶ **Example 10.** In the graph G of Figure 1, consider the permutation π first traversing all the top vertices from the left to the right and then all the bottom vertices from the left to the right. Let V_i be the set of all the top vertices (denoted by S in the picture). It is not hard to see that the largest induced $(V_i, V(G) \setminus V_i)$- matching of G^{V_i} is of size 1. The widths of the rest of the prefixes are also at most 1, So, $r(\pi) = 1$. Since the graph is connected, any permutation will have width at least one. So, we conclude that $lu(G) = 1$.

The parameter LU-MIM WIDTH can be considered as lying between existing parameters LMIM WIDTH and LSIM WIDTH. In particular, to compute the width of a prefix V_i for LMIM WIDTH, the edges having both ends in V_i are discarded along with the edges having both of their ends out of V_i. In Example 10, with $V_i = S$, only the edges between the top and the

bottom vertices remain, so the largest size of an induced $(V_i, V(G) \setminus V_i)$- matching of the resulting graph becomes 2. For LSIM WIDTH, no edges are discarded at all, so the width of V_i is the largest induced $(V_i, V(G) \setminus V_i)$-matching for the whole G.

It is clear that for any graph G, its LSIM WIDTH is smaller than or equal to its LU-MIM WIDTH which, in turn, is smaller than or equal to its LMIM WIDTH. For the latter two we can, in fact, demonstrate a class of graphs where LU-MIM WIDTH is bounded while LMIM WIDTH unbounded but we leave the exact relationship between the former two as an open question. We postpone to Section 6 a more detailed discussion of relationship between the parameters as well as justifying the need of the new parameter for bounding the size of OBDDs. The reason of this arrangement is that we need first to prove the main results of the paper so that we can refer to them for the purpose of the justification.

4 OBDD bounds in terms of LU-MIM width

In this section we establish upper and lower bounds on the size of OBDDs representing monotone two CNFs. The upper bound is the more interesting of these two because it is based on the following combinatorial statement.

▶ **Theorem 11.** *Let $U \subseteq V(G)$ such that $V = V(G) \setminus U$ is independent. Then $|\mathbf{TRACES}(U)| \leq n^{r+1}$ where $n = |V(G)|$ and r is the size of the largest induced (U, V)-matching.*

Before we provide a proof of Theorem 11, let us remark that if U is independent (that is G is a bipartite graph with U and V being its parts) then the statement follows from Sauer-Shelah lemma. This is just because, in this case, the size of the largest induced matching of G is exactly the VC-dimension of $\mathbf{TRACES}(U)$. Indeed, let $W = \{w_1, \ldots, w_q\}$ be a set of the largest size shattered by $\mathbf{TRACES}(U)$. Then we can identify subsets $U_1, \ldots, U_q$ such that $N(U_i) \cap W = \{w_i\}$ for $1 \leq i \leq q$. In particular, in each U_i we can identify a vertex u_i such that u_i is adjacent to w_i but not adjacent to any other vertex of W. Consequently, the edges $\{u_1, w_1\}, \ldots, \{u_q, w_q\}$ constitute an induced matching. Conversely, let $\{u_1, w_1\}, \ldots, \{u_q, w_q\}$ be an induced matching. Then the set $\{w_1, \ldots, w_q\}$ is shattered by neighborhoods of all possible subsets of $\{u_1, \ldots u_q\}$. Hence the VC dimension of $\mathbf{TRACES}(U)$ is at least q.

If U is not an independent set, the first part of the above reasoning does not work. Indeed, the vertices $u_1, \ldots, u_q$ extracted from $U_1, \ldots, U_q$ do not necessarily form an independent set and hence the resulting matching is not necessarily induced. We were unable to upgrade the above argument to prove Theorem 11 and hence we provide a self-contained proof.

Proof of Theorem 11.

▷ Claim 12. Let $S \subseteq U$ be an independent subset of U. Let $u \in S$. Suppose that $S \setminus \{u\}$ enables an induced (U, V)-matching while S does not. Then there is a subset $S' \subset S$ enabling an induced (U, V)-matching such that $N(S') \cap V = N(S) \cap V$.

Proof. By induction on $|S|$. For $|S| = 1$ the statement holds in a vacuous way. For each $u' \in S$ let $T(u') = (N(u') \cap V) \setminus (N(S \setminus \{u'\}) \cap V)$ be called the *individual trace* of u'. Suppose all the individual traces are non-empty. For all u' fix an arbitrary $v' \in T(u')$. Then $\{\{u', v'\} | u' \in S\}$ is an induced matching (recall that V is independent) contradicting our assumption. It follows that there is $u' \in S$ such that $T(u') = \emptyset$. But then $N(S) \cap V \subseteq N(S \setminus \{u'\}) \cap V$ and hence $N(S) \cap V = N(S \setminus \{u'\}) \cap V$. If $S' = S \setminus \{u\}$ enables an induced (U, V)-matching, we are done. Otherwise, apply the induction assumption to S'. ◁

▷ Claim 13. Let $S \subseteq U$ be an independent subset of U. Then there is $S' \subseteq S$ enabling an induced (U,V)-matching such that $N(S) \cap V = N(S') \cap V$.

Proof. Let q be the size of the largest subset of S enabling an induced (U,V)-matching. We proceed by induction on $|S| - q$. If it is zero then put $S' = S$. Otherwise, let S_0 be a subset of S of size q enabling an induced (U,V) matching and let $u \in S \setminus S_0$. By Claim 12, there is $S_1 \subset S_0 \cup \{u\}$ enabling an induced (U,V) matching such that $N(S_1) \cap V = N(S_0 \cup \{u\}) \cap V$.

Let $S_2 = S \setminus (S_0 \cup \{u\})$. Then

$$N(S_1 \cup S_2) \cap V = (N(S_1) \cap V) \cup (N(S_2) \cap V) = \\ (N(S_0 \cup \{u\}) \cap V) \cup (N(S_2) \cap V) = N(S_0 \cup \{u\} \cup S_2) \cap V = N(S) \cap V \quad (1)$$

Further on, $S_1 \cup S_2$ has a subset of size at least $|S_1|$ enabling an induced (U,V)-matching. But $|S_2 \cup S_1| - |S_1| = |S_2| = |S| - q - 1$. Apply the induction assumption to $S_1 \cup S_2$ to find a subset $S_3 \subseteq S_1 \cup S_2$ enabling an induced (U,V) matching such that $N(S_3) \cap V = N(S_1 \cup S_2) \cap V$. Since $S_1 \cup S_2 \subseteq S$, $S_3 \subseteq S$ and $N(S_3) \cap V = N(S) \cap V$ by (1), we put $S' = S_3$. □ ◁

By assumption an independent subset of U enabling an induced (U,V) matching is of size at most r. It follows from Claim 13 that $\mathbf{TRACES}(U) = \{N(S) \cap V | S \in \mathbf{ISET}(U), |S| \leq r\}$. Clearly the size of the right-hand set is upper bounded by the number of subsets of U of size at most r which is clearly upper bounded as claimed in the theorem. ◀

▶ **Theorem 14** (OBDD bounds). *For $\varphi = \varphi(G)$, $2^{lu(G)} \leq obdd(\varphi) \leq n^{O(lu(G))}$.*

Proof. Let $\pi = (v_1, \ldots, v_n)$ be a permutation of $V(G)$ witnessing $lu(G)$. Let V_i and r_i be as in Definition 9. By combination of Lemma 6 and Theorem 11, $|\mathbf{BF}(V_i)| \leq n^{r_i+1} \leq n^{lu(G)+1}$. The upper bound follows from the second statement of Proposition 3.

For the lower bound we assume now that $\pi = (v_1, \ldots, v_n)$ is an arbitrary permutation with the meaning of V_i and r_i retained. Furthermore, we assume that $(v_1, \ldots, v_i)$ is selected so that $r_i \geq lu(G)$ (such a prefix exists by definition of LU-MIM WIDTH). We are going to show that $|\mathbf{TRACES}(V_i)| \geq 2^{r_i}$. The lower bound will then follow from combination of Lemma 6 and the first statement of Proposition 3.

Let $U^* = \{u_1, \ldots, u_{r_i}\}$ be a subset of V_i enabling an induced $(V_i, V(G) \setminus V_i)$- matching of G^{V_i} of size r_i and let $M = \{\{u_1, v_1\}, \ldots, \{u_{r_i}, v_{r_i}\}\}$ be the edges of this matching. Let U_1, U_2 be two distinct subsets of U^*. We claim that $N(U_1) \cap (V(G) \setminus V_i) \neq N(U_2) \cap (V(G) \setminus V_i)$. Indeed, assume w.l.o.g. that there is $u_j \in U_1 \setminus U_2$. Then $v_j \in N(U_1) \cap (V(G) \setminus V_i)$ while $v_j \notin N(U_2) \cap (V(G) \setminus V_i)$. Thus 2^{r_i} subsets of U^* have pairwise distinct neighborhoods in V witnessing that $|\mathbf{TRACES}(V_i)| \geq 2^{r_i}$. ◀

5 No tighter bounds

We are now going to prove that the bounds in the statement of Theorem 14 are asymptotically best possible. This will imply that the quasypolynmial gap between the upper and lower bounds cannot be narrowed down. For the lower bound the proof will be straightforward. For the upper bound we will need a 'gadgeted' construction developed below.

▶ **Definition 15.** *Let $U = (u_1, \ldots, u_q)$, $V = (v_1, \ldots, v_q)$. The graph $SKEW(U,V)$ over vertices $\{u_1, \ldots, u_q, v_1, \ldots, v_q\}$ has the set of edges $\{\{u_i, v_j\} | i \leq j\}$.*

*Let $U_1, \ldots, U_p$ be mutually disjoint sequences of q elements. We define a graph G over $U_1 \cup \cdots \cup U_p$ (here we interpret U_i as sets) as follows. For each $1 \leq i \leq p-1$, $G[U_i \cup U_{i+1}]$ is $SKEW(U_i, U_{i+1})$. We call G a p,q-*path of skewed graphs*. We call $U_1, \ldots, U_p$ the* sequence of layers *of G and give them numbers $1, \ldots, p$ in the order listed.*

▶ **Definition 16.** *Let P be a path. The* 1*-subdivision of P is the graph obtained by introducing exactly one subdivision to each edge of P.*

▶ **Definition 17.** *Let $G^1, \ldots, G^r$ be p, q-paths of skewed graphs with respective sequences $(U_1^1 \ldots, U_p^1), \ldots, (U_1^r, \ldots, U_p^r)$ of layers. Connect the vertices of each $U_i^1 + \cdots + U_i^r$ into a path P in the order specified and* 1*-subdivide the resulting path. Let G be the resulting graph. We call G a p, q, r-*grid of skewed graphs *(we may omit the parameters if they are not relevant in the context).*

The vertices $V(G^1) \cup \cdots \cup V(G^r)$ are referred to as the main vertices *and the vertices introduced by the* 1*-subdivision are the* auxiliary vertices*. The subdivided paths are referred to as the* layers *of G with the i-th layer being the one containing $U_i^1, \ldots, U_i^r$ as the main vertices. Let us enumerate the main vertices of each layer i as in the sequence $U_i^1 + \ldots, +U_i^r$ starting from* 1*. The number each vertex receives is the* coordinate *of this vertex.*

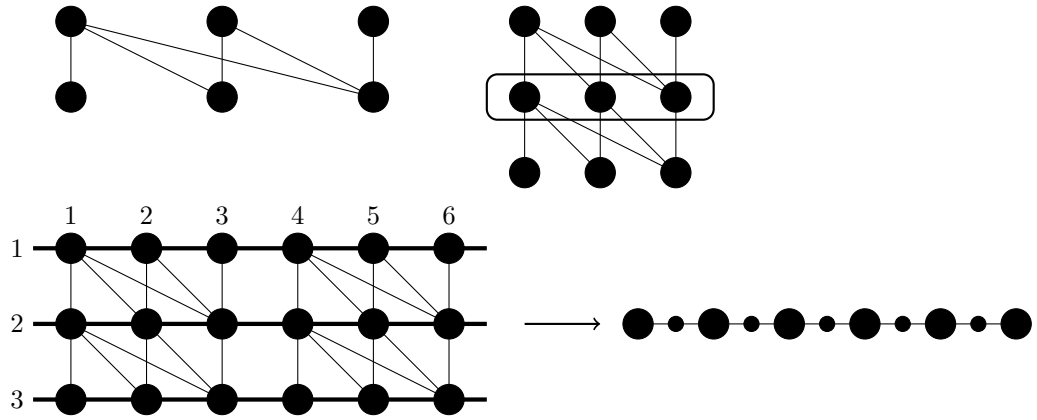


Figure 2 A grid of skewed graphs.

Figure 2 demonstrates a grid of skewed graphs. The top-left graph is $SKEW(U, V)$ where U is the sequence of three vertices on the top enumerated from the left to the right and V is the respective sequence of the bottom vertices. The graph on the top-right is a $3, 3$-path of skewed graphs. The graph has three layers enumerated from the top to the bottom. The vertices of the second layer are surrounded by the oval. The graph at the bottom-left is the $3, 3, 2$-grid of skewed graphs. For the sake of a better visualization, the auxiliary vertices are not shown and the layers are denoted by thick lines, the meaning of a thick line is specified on the bottom right of the picture. The grid has three layers and the coordinates of the main vertices range from 1 to 6 as specified in the picture.

▶ **Definition 18.** *Let G be an p, q, r-grid of skewed graphs. Let U be a set of main vertices one of each coordinate and none belonging to the last layer. For each vertex $u_1 \in U$ let u_2 be the vertex of the same coordinate lying at the next layer. Let V be the set of all vertices u_2. Let H be the subgraph of G induced by $U \cup V$. We call H a* horizontal *subgraph of G. We call U, V the* top and bottom forming sets *of H. Let M be the matching consisting of all the edges $\{u_1, u_2\}$ as above. We call M the* core matching *of H. $U \cup V$ is partitioned into r* main intervals *$1, \ldots, r$ where vertices of the i-th main interval are those having coordinates $(i-1) * q + 1, \ldots, i * q$.*

▶ **Lemma 19.** *With the notation as in Definition 18, both $|\mathbf{TRACES}_H(U)| \geq (q+1)^r$ and $|\mathbf{TRACES}_H(V)| \geq (q+1)^r$*

Proof. We prove only the first statement, the second is symmetric.

Let $H_1, \ldots, H_r$ be the subgraphs of H induced by the respective main intervals $1, \ldots, r$. For each H_i denote $V(H_i) \cap U$ and $V(H_i) \cap V$ by U_i and V_i, respectively.

It is not hard to see that H is the disjoint union of $H_1, \ldots, H_r$, hence $|\mathbf{TRACES}_H(U)| = \prod_{i=1}^{r} |\mathbf{TRACES}_{H_i}(U_i)|$. It is thus sufficient to prove that for each i, $|\mathbf{TRACES}_{H_i}(U_i)| \geq q+1$. W.l.o.g. we only prove that $|\mathbf{TRACES}_{H_1}(U_1)| \geq q+1$.

For $U' \subseteq U_1$, let $first(U')$ be the vertex $u' \in U'$ located at the layer having the largest number (among the vertices of U'), and, among those vertices of U' located at the layer, having the smallest coordinate. Let $u_1 = first(U_1)$ and for $2 \leq i \leq q$, $u_i = first(U_1 \setminus \{u_1, \ldots, u_{i-1}\})$. Let $v_1, \ldots, v_q$ be the other ends of the edges of M (the core matching of H) incident to $u_1, \ldots, u_q$, respectively. Observe that H has no edge $\{u_i, v_j\}$ such that $i > j$. Indeed, otherwise, either the layer of u_j is smaller than the layer of u_i or the coordinate of u_j is larger than the coordinate of u_i, both cases contradict the choice of vertices $u_1, \ldots, u_q$.

Consider the sets $W_1, \ldots W_{q+1}$ such that $W_{q+1} = \emptyset$ and $W_j = \{u_j\}$ for $1 \leq j \leq q$. It follows that for each $1 \leq j \leq q$, $v_j \in N(W_j) \cap V_1$ and $v_j \notin N(W_k) \cap V_1$ for $k > j$. It follows that the sets $N(W_1) \cap V_1, \ldots, N(W_{q+1}) \cap V_1$ are all distinct thus confirming that $|\mathbf{TRACES}_{H_1}(U_1)| \geq q+1$. ◀

▶ **Lemma 20.** *Let G be a p, q, r-grid of skewed graphs where $p > 1, q > 1, r \geq 1$, and $p = 2 * r\lceil \log q \rceil$. Let $n = V(G)$. Then for $\varphi = \varphi(G)$, $obdd(\varphi) \geq n^{r/2}$ for a fixed r and a sufficiently large n.*

Proof. We prove the q^r lower bound instead of $n^{r/2}$. Indeed $n^{r/2} \leq (2 * q * p * r)^{r/2} = q^{r/2} * (4r^2\lceil \log q \rceil)^{r/2} \leq q^r$ for a fixed r and a sufficiently large q. We consider an arbitrary permutation π and show existence of a prefix π' such that $|\mathbf{TRACES}(\pi')| \geq q^r$. The lemma will then follow from the combination of the first statement of Proposition 3 and Lemma 6.

Let π' be the shortest prefix of π containing all the vertices of some layer x. Assume existence of a layer y none of which vertices are contained in π'. Assume that $y > x$. For each coordinate i, specify main vertices u_i, v_i both having coordinate i with the layer of v_i being the next after the layer of u_i and such that $u_i \in \pi'$ while $v_i \notin \pi'$. To see that such vertices exist, start from the main vertex with coordinate i at layer x and iteratively move down. Since the respective vertex of coordinate i at y is not in π', the required vertices u_i, v_i will eventually be found. The set $\{u_1, \ldots, u_{qr}, v_1, \ldots, v_{qr}\}$ induce a horizontal subgraph H of G with $U = \{u_1, \ldots, u_{qr}\}$ being the top set. Then $|\mathbf{TRACES}(\pi')| \geq |\mathbf{TRACES}_H(U)| \geq (q+1)^r$, the last inequality follows from Lemma 19.

If $y < x$, the reasoning is symmetric and we use the second statement of Lemma 19 rather than the first one. It remains to assume that at least one vertex of each layer of G is contained in π'. Remove from π' the last vertex and let π^* be the resulting prefix. By definition of π', in each layer of G there is at least one vertex inside π^* and at least one vertex outside π^*. Since layers induce connected subgraphs of G, we can identify edges $\{u_i, v_i\}$ of G for $1 \leq i \leq p$ with u_i, v_i located at layer i, u_i is contained in π^* while v_i is not. We notice that the edges $\{u_i, v_i\}$ with odd indices form an induced matching. Indeed, in G two vertices are adjacent only if they are in the same layer or in consecutive layers. Let U be the set of vertices u_i with i being odd. Applying the argument as in the lower bound proof for Theorem 14, we observe that the neighborhoods of the subsets of U in $V(G) \setminus \pi^*$ are pairwise distinct. Taking into account the definition of p and that $|U| = p/2$ by construction, we conclude that $\mathbf{TRACES}(\pi^*) \geq 2^{|U|} = 2^{p/2} \geq q^r$. ◀

▶ **Lemma 21.** *With the notation as in Lemma 20, $r \leq lu(G) \leq r+2$.*

Proof. For the lower bound we argue as in Lemma 20. Recall that for an arbitrary permutation π we considered two cases. In the first case we observed existence of a prefix π' such that there is a horizontal subgraph H of G with all vertices of the top forming set contained in π' and all vertices of the bottom forming set being outside of π' (or vice versa). For each main interval take one edge of the core matching whose vertex coordinates belong to this interval. These edges, taken together constitute an induced matching of G of size r.

If prefix π' as above does not exist then there is a prefix π^* such that for each layer $1 \leq i \leq p$ there is an edge $\{u_i, v_i\}$ with $u_i \in \pi^*$ and $v_i \notin \pi^*$. As we have observed the edges with odd indices comprise an induced matching of G of size at least $r \log q > r$.

For the upper bound, we consider a permutation π where vertices occur layer by layer: first layer 1, then layer 2 and so on. Within each layer the vertices occur along the path induced by the layer starting from the main vertex with coordinate 1.

Consider a prefix π' of π. Let x be the largest layer number (some of) whose vertices are contained in π'. By definition of π all the vertices whose layer numbers are smaller than x belong to π'. It follows that the edges between π' and $V(G) \setminus \pi'$ belong to one of the following categories.

1. Edges between layer x and layer $x+1$. Suppose that π' contains vertices of layer x laying in intervals $1, \ldots r'$. Then this category of edges can contribute at most r' edges to an induced matching of $G^{\pi'}$.
2. Edges between layer $x-1$ and layer x. This category of edges can contribute at most $r - r' + 1$ edges to the induced matching (the extra one is on the account that not all vertices of interval r' and layer x may be present in π') so there may be an edge of vertices of interval r' between layers $x-1$ and x contributing to the considered induced matching.
3. Edges with both ends in layer x. Since π' contains an initial fragment of the path of layer x, there may be at most one such an edge.

Summing up the above three items, we conclude that the size of induced matching of $G^{\pi'}$ cannot be greater than $r + 2$. ◀

▶ **Theorem 22** (best possible bounds). *For every fixed $r \geq 1$ there are infinite classes $\mathcal{G}_1$ and $\mathcal{G}_2$ of graphs of* LU-MIM WIDTH $\Theta(r)$ *and such that for each $G_1 \in \mathcal{G}_1$, $obdd(\varphi(G_1)) \leq 2^{O(r)}$ while for each $G_2 \in \mathcal{G}_2$, $obdd(\varphi(G_2)) \geq n^{\Omega(r)}$.*

Proof. Let $\mathcal{G}_1$ be the set of all $p \times r$ grids for a sufficiently large p. Each graph of this class has pathwidth of $\Theta(r)$ and hence the OBDD size is at most $2^{O(r)}$ by [3]. Because of the bounded degree, the pathwidth and the LU-MIM WIDTH of graphs in $\mathcal{G}_1$ are linearly related. Hence, we conclude that for each $G_1 \in \mathcal{G}_1$, $obdd(\varphi(G_1)) = 2^{O(lu(G))}$. Let $\mathcal{G}_2$ be the class of p, q, r grids for a sufficiently large q and $p = 2 * r\lceil \log q \rceil$. The required properties are immediate from the combination of Lemma 20 and Lemma 21. ◀

6 Why is the new parameter needed

In this section we justify the need for the new parameter of LU-MIM WIDTH. In particular, we explain why we cannot use two existing closely related parameters: LMIM WIDTH and LSIM WIDTH. For the sake of completeness, let us define the latter two parameters.

▶ **Definition 23.** *Let $\pi = (v_1, \ldots, v_n)$ be a permutation of $V(G)$. Denote $\{v_1, \ldots, v_i\}$ by V_i. Let x_i be the largest size of an induced matching of $G[V_i, V(G) \setminus V_i]$ which is the graph induced by the edges between V_i and $V(G) \setminus V_i$. Let y_i be the largest size of an induced $(V_i, V(G) \setminus V_i)$-matching of G. Let $x(\pi)$ be the maximum of all x_i and let $y(\pi)$ be the maximum of all y_i. The Linear Maximum Induced Matching Width (*LMIM WIDTH*) of G*

denoted by $lmimw(G)$ is the minimum $x(\pi)$ over all permutations π of $V(G)$. The Linear Special Induced Matching Width (LSIM WIDTH) of G denoted by $lsimw(G)$ is the minimum $y(\pi)$ over all the permutations π of $V(G)$.

The parameter LMIM WIDTH cannot be used for our purposes because it does not capture the lower bound for OBDDs representing monotone 2-CNFs. In particular, below we demonstrate an infinite class of graphs having LMIM WIDTH of order of the square root of the number of vertices whose corresponding CNFs can be represented by polynomial size OBDDs.

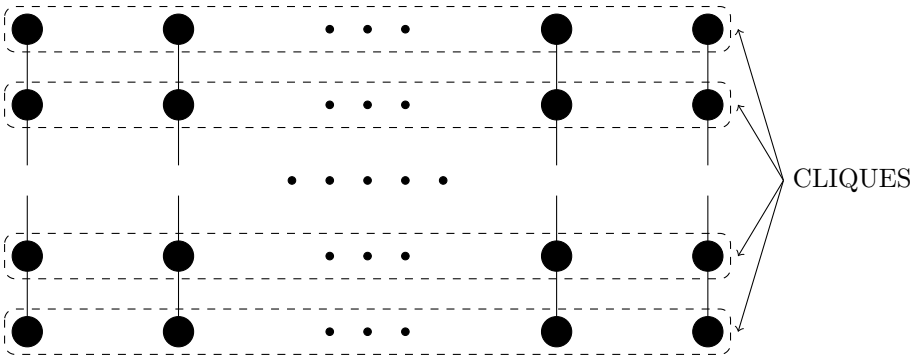


Figure 3 Schematic illustration of graphs H_n.

▶ **Theorem 24.** *For each integer $r \geq 2$, there is an infinite class of graphs H_r of $n = r^2$ vertices such that $lu(H_r) = 2$ (and hence $\varphi(H_r)$ can be represented by an OBDD of size at most $n^{O(1)}$ by Theorem 14) while $lmimw(H_r) \geq (r-1)/2$.*

Proof. $V(H_r)$ consists of disjoint union of sets $V_1, \ldots, V_r$ of r vertices each. Each V_i is a clique in H_r. Denote the vertices of each V_i by $v_{i,1}, \ldots, v_{i,r}$. The graph H_r has paths $v_{1,i}, \ldots, v_{r,i}$ for each $1 \leq i \leq r$. $E(H_r)$ contains no other edges besides those specified above. Figure 3 schematically illustrates the graphs H_r.

To demonstrate that $lu(H_r)$ is small, consider the permutation of $V(H_r)$ by the alphabetic ordering of their indices, that is, $v_{1,1}, \ldots, v_{1,r}, \ldots, v_{r,1}, \ldots, v_{r,r}$. Let V' be the set of vertices of a prefix of this permutation. Let $1 \leq q \leq r$ be such that $V_q \cap V' \neq \emptyset$ while for each $q < i \leq r$, $V_i \cap V' = \emptyset$. It follows that for each $1 \leq i < q$, $V_i \subseteq V'$. Hence, by construction, any edge connecting V' to $V(H_r) \setminus V'$ has an end either in V_q or in V_{q-1}. Thus for any three edges between V' and $V(H_r) \setminus V'$ either two of them have an end in V_q or two of them have an end in V_{q-1}. In both cases these ends are connected by an edge with both ends lying in V' and hence such edges cannot constitute an induced matching of $H_r^{V'}$. We conclude that the largest possible size of the such an induced matching is 2. It follows from Theorem 14 that $\varphi(H_r)$ can be represented by an OBDD of size upper bounded by $n^{O(1)}$.

Let us know establish an $\Omega(r) = \Omega(n^{1/2})$ lower bound on $lmimw(H_r)$. For vertices $v_{i,j}$ of H_r let us call their first coordinates *rows* and their second coordinates *columns*. Let π be an arbitrary permutation of $V(H_r)$. Let π_0 be the longest prefix of π that does not contain vertices with all the row coordinates and does not contain vertices with all the column coordinates. Since this is already not the case for the immediate successor of π_0, it is either that π_0 contains vertices with $r-1$ row coordinates or vertices with $r-1$ column coordinates. Assume the former. Then there is a set I of $r-1$ rows i such that π_0 contains some $v_{i,j}$. By assumption, for each $i \in I$, there is some q such that $v_{i,q} \notin \pi_0$. Since each V_i is connected we can identify, for each $i \in I$ an edge $\{v_{i,j_1}, v_{i,j_2}\}$ such that one end of this edge is in π_0 while

the other end is outside. At least half of such edges have the same parity of the row. Let M be a such a subset of edges. By definition of H_r vertices with the same row parity are not adjacent hence this matching is induced in H_r and of size at least $(r-1)/2$ by definition.

It remains to assume that there is a set I of $r-1$ columns j such that there is a vertex $v_{i,j} \in \pi_0$. By assumption, at least one vertex of $v_{1,j}, \ldots, v_{r,j}$ does not belong to π_0. As $v_{1,j}, \ldots, v_{r,j}$ induce a path, there is an edge $\{v_{i,j}, v_{i+1,j}\}$ such that one end belong to π_0 while the other end is outside. Let E' be a set of such edges one per column of I. For an edge $\{v_{i,j}, v_{i+1,j}\}$ of E' we call its end that belongs to π_0 the *inner end* and the other one the *outer end*. We call the edge *even* if the row of the inner end is even and *odd* otherwise. Clearly at least $(r-1)/2$ edges of E' have the same parity. Assume without loss of generality that these are even edges. It remains to demonstrate that there are no distinct columns j_1 and j_2. such that the inner end of the edge of E' of column j_1 is adjacent to the outer end of the edge corresponding to j_2. Since the columns are different the adjacency may be only because the adjacent ends belong to the same clique V_i. But this means that the row of the inner end of j_2 is odd, a contradiction. ◀

Regarding LSIM WIDTH, the situation is opposite: LSIM WIDTH cannot represent the OBDD upper bound. In particular, we present below a class of graphs whose LSIM WIDTH is at most 3 while the LU-MIM WIDTH is lower bounded the number of vertices to the power of 1/3. Hence, the size of the corresponding OBDDs is exponential in the number of vertices (in a positive power). We conclude that LSIM WIDTH cannot be used for representation of the OBDD upper bound for monotone 2-CNFs.

▶ **Definition 25.** *For each integer $r \geq 2$, we define the graph X_r of $n = 2r^3$ vertices as follows. Let $L_1, \ldots, L_{2r}$ be mutually disjoint sets of r^2 vertices in each and call the sets* layers*. $V(X_r) = L_1 \cup \cdots \cup L_{2r}$.*

For the purpose of introducing edges, each L_i is arbitrarily partitioned into sets $L_{i,1}, \ldots, L_{i,r}$ of r vertices in each. These sets are called sublayers *of layer i. The vertices of each $L_{i,j}$ are arbitrarily enumerated.*

The edges of X_r are the following.

1. *For each $1 \leq i \leq 2r-1$, for each $1 \leq j \leq r$ and for each $1 \leq k \leq r$, introduce an edge between vertex number k of $L_{i,j}$ and vertex number k of $L_{i+1,j}$. We call these edges* inter-layer *ones.*
2. *For each odd i for every $1 \leq j < k \leq r$ introduce an edge between each vertex of $L_{i,j}$ and each vertex of $L_{i,k}$.*
3. *For each even i and each $1 \leq j \leq r$, make $L_{i,j}$ into a clique.*

▶ **Lemma 26.** *$lsimw(X_r) \leq 3$.*

Proof. Consider the following permutation π of X_r. The vertices are traversed layer by layer, first L_1 then L_2 then L_3 and so on. Within each L_i first vertices of $L_{i,1}$ are traversed then of $L_{i,2}$, and so on. Within each $L_{i,j}$ vertices are traversed by the increasing order of the numbers assigned to them.

Consider an arbitrary prefix V' of this permutation. Let q be the largest number such that $L_q \cap V' \neq \emptyset$. It follows that for each $1 \leq i \leq q-1$ $L_i \subseteq V'$. Consequently, by construction, the edges between V' and $V(X_r) \setminus V'$ may be divided into the following three categories.

1. Edges between L_q and L_{q+1}. Then any two such edges will have an edge between their ends. Indeed, any such an edge connects an odd layer with an even layer. Let us call the end in the odd layer the *odd end* and the end in the even layer the *even end*. Note that both odd ends belong to the same sublayer if and only if both even ends do. If both even

ends belong to the same sublayer then they are adjacent by construction. Otherwise, both odd ends belong to different sublayers and, again, are adjacent by construction.

2. Edges between L_{q-1} and L_q. The same principle applies that any two edges must have adjacent ends.
3. Edges inside L_q. If q is odd then the ends of any edge e belong to different sublayers. By construction $N(e) \cap L_q = L_q \setminus e$ hence any two edges have adjacent ends. In case q is even, there are two cases two consider. The first is when each sublayer of L_q is either a subset of V' or a subset of $V(X_r) \setminus V'$. In this case there are no edges between V' and $V(X_r) \setminus V'$ with both ends in L_q. Otherwise, there is precisely one sublayer of L_q that is in part in V' and in part outside of V'. But since this sublayer induced a clique of X_r clearly any two edges of this category wil have adjacent ends.

If we take arbitrary four edges between V' and $V(X_r) \setminus V'$ then two of them will get to the same category (by the pigeonhole principle because there are three categories) and, as specified above there is an edge between their ends. Hence the width of the prefix is at most 3. ◀

▶ **Lemma 27.** $lu(X_r) \geq r$.

Proof. Let π be an arbitrary permutation of $V(X_r)$.

▷ Claim 28. Suppose that π has a prefix π' such that for each odd layer L_i, $L_i \cap \pi' \neq \emptyset$ and $L_i \setminus \pi' \neq \emptyset$. Then $X_r^{\pi'}$ has an induced matching of size r.

Proof. As each odd L_i induces a connected subgraph we can identity an edge in $X_r[L_i]$ with one end in π' the other edge out of π'. Let E' be the set of such edges of all r odd layers. By construction they form an induced matching. ◁

▷ Claim 29. Assume that π has a prefix π' such that for each $1 \leq i \leq r$ there is an interlayer edge whose ends belong to sublayer i with the even end in π' and the odd end outside π'. Let E' be a set of such n edges. Then they form an induced matching of $X_r^{\pi'}$.

Proof. Indeed, let $\{u_1, v_1\}$ and $\{u_2, v_2\}$ be two such edges with u_1 and u_2 being the even ends. There is no edge between u_1 and u_2 as vertices of even layers belonging to different sublayers are not adjacent by construction. As vertices outside of π' form an independent set by definition of $H_n^{\pi'}$ v_1 and v_2 are not adjacent. Now u_1 and v_2 may be adjacent only if they belong to the same sublayer which is not the case, likewise for u_2 and v_1. ◁

It remains to assume that the cases as in the above two claims do not hold. Let π_0 be the shortest prefix such that there is an odd layer L_i with $L_i \subseteq \pi_0$ (*the full layer*). Then there is another odd layer L_j such that $L_j \cap \pi' = \emptyset$ (*the empty layer*). Indeed, otherwise, let π_1 be the immediate predecessor of π_0. The only difference of π_1 from π_0 is that one vertex of L_i is outside of π_1 simply due to the minimality of π_0. By assumption about π_0 each odd layer has a vertex inside π_1. By minimality of π_0 each odd layer has a vertex outside π_1. This is exactly the situation as in the Claim 28 in contradiction to our assumption.

Next, to avoid the premises of Claim 29 to apply, we identify $1 \leq k \leq r$ such that there is no inter-layer edge between π' and $V(G) \setminus \pi'$ with the both ends in sublayer k, the end in π' being the even one. We can assume w.l.o.g. that $k = 1$.

For each $1 \leq x \leq r$ there is an interlayer edge whose nodes have number x in layer 1, one end inside π' the other end outside π'. Indeed, assume w.l.o.g. that $j > i$ (recall that i and j are the numbers of the full and empty layers respectively). Let P be the path formed of interlayer edges of sublayer 1 whose ends have number x consisting of vertices at layers

$i, i+1, \ldots, j$. The vertex at layer i is in π', the vertex at layer j is outside π', hence one of the edges of P ought to be as desired. Let E' be the set of r such edges. By our assumption each edge of E' has its odd end inside π' and the even end outside π'. We are going to show that the edges of E' form an induced matching of $X_r^{\pi'}$.

Let $\{u_1, v_1\}$ and $\{u_2, v_2\}$ be two edges of E', u_1 and u_2 being the ends inside π', v_1 and v_2 being the ends outside π'. Now v_1 and v_2 are not adjacent by definition of an upper graph. The vertices u_1 and u_2 are not adjacent because, by construction two different vertices of the same sublayer of odd layers are not adjacent. Finally u_1 and v_2 as well as u_2 and v_1 are vertices of layers of different parity lying in the same sublayers but having different numbers. Again, by construction, such vertices cannot be adjacent. ◀

▶ **Theorem 30.** *There is no function f such that for any graph G, $obdd(\varphi(G)) \leq n^{f(lsimw(G))}$.*

Proof. Consider the graphs X_r. By Lemma 27, $lu(X_r) \geq r = (n/2)^{1/3}$ and hence, by Theorem 14, $obdd(\varphi(X_r) \geq 2^{n^{\Omega(1)}}$.

Clearly $obdd(\varphi(X_r))$ cannot be upper-bounded by any polynomial function of n. On the other hand, by Lemma 26, whatever function f we take, $n^{f(lsimw(X_n))}$ is upper-bounded by $n^{O(c)}$ where c is the maximum of $f(1)$, $f(2)$, and $f(3)$. ◀

7 Future research

In this section we discuss several interesting open questions related to representation of monotone 2-CNFs by circuit models more powerful than OBDDs. A natural question in this direction is to consider Nondeterministic Read-Once Branching Programs (1-NBPs) instead of OBDDs, For example, is it true that the size of 1-NBP representing a monotone 2-CNF $\varphi = \varphi(G)$ is lower bounded by $2^{\Omega(lu(G))}$?

Similarly to MIM WIDTH and SIM WIDTH, it is possible to formulate the 'non-linear' version of LU-MIM WIDTH in terms of the branch decompositions rather than permutations. It is interesting to investigate whether the size of Decomposable Negation Normal Forms (DNNFs) or its restricted classes such as Deterministic DNNFs representing monotone 2-CNFs can be captured by this non-linear parameter. We conjecture that the resulting non-linear parameter 'captures' the size of Structural Deterministic DNNFs but for more general models the situation is unclear and is likely to depend on the situation with 1-NBPs. Our belief relies on a plausible analogy with the bounded degree case where pathwidth captures the size of 1-NBPs while its non-linear counterpart (that is treewidth) captures the size of DNNFs [1].

Finally, a natural question arising from the results of this paper is to capture the size of OBDDs on monotone CNFs of higher arity. One possibility to achieve this might be through a concise generalization of LU-MIM WIDTH to hypergraphs.

References

1 Antoine Amarilli, Florent Capelli, Mikaël Monet, and Pierre Senellart. Connecting knowledge compilation classes and width parameters. *Theory Comput. Syst.*, 64(5):861–914, 2020.

2 Adnan Darwiche. Decomposable negation normal form. *J. ACM*, 48(4):608–647, 2001.

3 Andrea Ferrara, Guoqiang Pan, and Moshe Y. Vardi. Treewidth in verification: Local vs. global. In *Logic for Programming, Artificial Intelligence, and Reasoning, 12th International Conference (LPAR)*, pages 489–503, 2005.

4 Lars Jaffke, O-joung Kwon, Torstein J. F. Strømme, and Jan Arne Telle. Mim-width III. graph powers and generalized distance domination problems. *Theor. Comput. Sci.*, 796:216–236, 2019.

5 Lars Jaffke, O-joung Kwon, and Jan Arne Telle. Mim-width i. induced path problems. *Discret. Appl. Math.*, 278:153–168, 2020.

6 Lars Jaffke, O-joung Kwon, and Jan Arne Telle. Mim-width II. the feedback vertex set problem. *Algorithmica*, 82(1):118–145, 2020.

7 Stasys Jukna. *Extremal Combinatorics: With Applications in Computer Science*. Springer, 2011.

8 Dong Yeap Kang, O-joung Kwon, Torstein J. F. Strømme, and Jan Arne Telle. A width parameter useful for chordal and co-comparability graphs. *Theor. Comput. Sci.*, 704:1–17, 2017.

9 Stefan Mengel. Lower bounds on the mim-width of some graph classes. *Discret. Appl. Math.*, 248:28–32, 2018.

10 Igor Razgon. On the read-once property of branching programs and cnfs of bounded treewidth. *Algorithmica*, 75(2):277–294, 2016.

11 Igor Razgon. On oblivious branching programs with bounded repetition that cannot efficiently compute cnfs of bounded treewidth. *Theory Comput. Syst.*, 61(3):755–776, 2017.

12 Sigve Hortemo Sæther, Jan Arne Telle, and Martin Vatshelle. Solving #SAT and MAXSAT by dynamic programming. *J. Artif. Intell. Res.*, 54:59–82, 2015.

13 Martin Vatshelle. *New width parameters of graphs*. PhD thesis, Department of Informatics, University of Bergen, 2012.

14 Ingo Wegener. *Branching Programs and Binary Decision Diagrams*. SIAM Monographs on Discrete Mathematics and applications, 2000.

The PACE 2021 Parameterized Algorithms and Computational Experiments Challenge: Cluster Editing

Leon Kellerhals
Algorithmics and Computational Complexity, Faculty IV, Technische Universität Berlin, Germany

Tomohiro Koana
Algorithmics and Computational Complexity, Faculty IV, Technische Universität Berlin, Germany

André Nichterlein
Algorithmics and Computational Complexity, Faculty IV, Technische Universität Berlin, Germany

Philipp Zschoche
Algorithmics and Computational Complexity, Faculty IV, Technische Universität Berlin, Germany

Abstract

The Parameterized Algorithms and Computational Experiments challenge (PACE) 2021 was devoted to engineer algorithms solving the NP-hard Cluster Editing problem, also known as Correlation Clustering: Given an undirected graph the task is to compute a minimum number of edges to insert or remove in a way that the resulting graph is a cluster graph, that is, a graph in which each connected component is a clique.

Altogether 67 participants from 21 teams, 11 countries, and 3 continents submitted their implementations to the competition. In this report, we describe the setup of the challenge, the selection of benchmark instances, and the ranking of the participating teams. We also briefly discuss the approaches used in the submitted solvers.

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

Keywords and phrases Correlation Clustering, Cluster Editing, Algorithm Engineering, FPT, Kernelization, Heuristics

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.26

Supplementary Material *Software (Source Code)*: `https://github.com/PACE-challenge/Cluster-Editing-PACE-2021-instances`
archived at `swh:1:dir:502489a3535fb499b9bf59fc6ed185fb9a043c2d`

Funding This work has been partially supported by the DFG project FPTinP (NI 369/18). The publication of the proceedings of PACE 2021 has been supported by the Algorithmics and Computational Complexity group at Technische Universität Berlin.
Tomohiro Koana: Supported by the DFG project DiPa (NI 369/21)

Acknowledgements The PACE challenge was supported by Networks [1] and Technische Universität Berlin. The prize money (€4000) was generously provided by the *Networks* [1], 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, led by Szymon Wasik, and especially to Jan Badura and Artur Laskowski for the fruitful collaboration and for hosting the competition at the `optil.io` online judge system. We thank Aleksander Figiel (Technische Universität Berlin) for supporting us with scripts in the data collection process.

1 Introduction

The Parameterized Algorithms and Computational Experiments Challenge (PACE) is an annually held algorithm engineering competition conceived in Fall 2015 to deepen the relationship between parameterized algorithmics and practice. It aims to:

1. Bridge the divide between the theory of algorithm design and the practice of algorithm engineering.
2. Inspire new theoretical developments.
3. Investigate the competitiveness of theoretical algorithms from the field of parameterized complexity analysis and related fields in practice.
4. Produce universally accessible libraries of implementations and repositories of benchmark instances.
5. Encourage the dissemination of these findings in scientific papers.

In each of the five prior iterations [27, 28, 21, 32, 46] as well as this iteration, participants were asked to provide implementations for one or two specifically chosen problems which provide optimal as well as close to optimal solutions on a given set of selected instances in an appropriate amount of time. In the previous iterations, PACE tackled the problems Treewidth [27, 28], Feedback Vertex Set [27], Minimum Fill-in [28], Steiner Tree [21], Vertex Cover [32], Hypertree width [32], and Treedepth [46]. These challenges have had a significant impact on the research community According to Google Scholar, the previous PACE reports are cited more than 145 times altogether. Moreover, research articles based on concrete implementations competing in previous editions of PACE were published in conferences such as ALENEX, ESA, SEA, and WADS.

In this article, we report on the sixth iteration of PACE. The problem chosen for PACE 2021 is Cluster Editing, also known as Correlation Clustering (see Section 2 for the definition and overview). The challenge featured three tracks. In the *exact track* the goal was to compute optimal solutions for as many instances as possible with a 30-minute time limit per instance. In the *heuristic track* the goal was to compute valid solutions that are as close as possible to being optimal within 10 minutes per instance. In the (new) *kernel track* the goal was twofold: first, to compute an equivalent instance (referred to as kernel) that is as small as possible and, second, to lift valid (but not necessarily optimal) solutions for the kernel to valid solutions for the original instance; we refer to Section 3.1 for a more detailed description of the tracks and their aims.

The PACE 2021 challenge was announced on 22nd October 2020, tracks were specified on 19th November. On 16th December the public instances were made available. From 28th March 2021 on, the participants could test solutions on the public instances via the `optil.io` platform, which provided also a provisional ranking. The final version of the submissions was due on 1st June 2021. Afterwards, the submissions were evaluated on the public as well as a set of hidden instances (see Section 3.2 for details). The results were announced on 16th July 2021. The award ceremony took place during the International Symposium on Parameterized and Exact Computation (IPEC 2021) which was supposed to take place in Lisbon, but due to the pandemic crisis was held online. After the debut with PACE 2020, the current iteration is the second in which short descriptions of the top four solvers in each track are contained as standalone documents in the proceedings of IPEC.

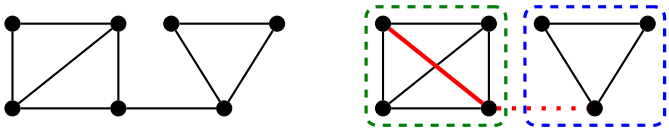


Figure 1 *Left:* An exemplary input graph. *Right:* Two edge modifications (deleting the red dotted edge and adding the thick red edge) suffice to obtain this cluster graph from the input graph. The two clusters are indicated by dashed boxes.

2 Cluster Editing

Graph-based data clustering has numerous applications and there are many approaches to cluster a given graph [58]. Cluster Editing, also known as Correlation Clustering, follows the graph modification approach [4, 9, 59]: Given an undirected graph, the task is to find a minimum-cardinality set of edges to insert or remove in a way that the resulting graph is a cluster graph – a graph where every connected component is a complete graph (called a clique) – see Figure 1 for an example. Herein, the assumption is that a cluster graph gives an ideal clustering: each cluster is maximally connected and no edge exists between two clusters. The graph modification approach lets us find a cluster graph "closest" to the input, that is, a best clustering under the parsimony criterion. One important advantage of this approach is that the number of clusters is not required to be part of the input but is determined implicitly by the input graph. The application fields of Cluster Editing include bioinformatics [9], data mining [4], and psychology [64].

For a given graph $G = (V, E)$ we call a set $S \subseteq \binom{V}{2}$ of vertex pairs a *cluster editing set* if $(V, E \Delta S)$ is a cluster graph, where Δ denotes the symmetric difference.

A graph is a cluster graph if and only if it does not contain a P_3 as an induced subgraph. This characterization gives rise to a simple integer linear programming formulation [41] as well as the following branching strategy: For every induced P_3, add the missing edge to make it a clique, or remove one of the two edges of the P_3 [22]. This results in an algorithm with running time $O(3^k \cdot |V|^3)$, where k is the size of the cluster editing set. A first improvement of this simple branch-and-bound algorithm is due to Gramm et al. [39]; among other results they showed that Cluster Editing is solvable in $O(2.27^k \cdot |V|^3)$ time. Their algorithm combines the P_3 branching strategy with heavy case distinction. The to this date fastest fixed-parameter algorithm with respect to k runs in $O(1.62^k + n + m)$ time [17]. This algorithm uses the so-called *merge branching* technique: When faced with a P_3 induced by the vertices u, v, w, one decides whether or not u and v will end up in the same cluster, and correspondingly merges u and v into a single vertex uv or deletes the edge $\{u, v\}$, respectively. Note that for the merge step one has to introduce edge weights for the edges incident to uv and deal with the special case of weight-0 edges. We remark that all solvers submitted to the exact track solve an integer program or use a branch-and-bound strategy at the heart of their algorithm.

Among many further studies in parameterized algorithmics [11, 18, 34, 44, 48] it was shown that an algorithm with running time subexponential in k, the number of vertices, or the number of edges would refute the exponential time hypothesis (ETH) [45]. Furthermore, Cluster Editing admits polynomial-size kernelizations. Studies in this direction were initialized by Gramm et al. [39], who provided a kernel with $O(k^2)$ vertices. Over the next years the kernel size was improved to $24k$ vertices [33], $4k$ vertices [42], and finally $2k$ vertices [23, 24].

Observe that any cluster editing set is guaranteed to contain at least one edge for every disjoint P_3, so it is natural to ask whether Cluster Editing remains fixed-parameter tractable for the number of edges *above guarantee*. In this line of thought it was shown that Cluster Editing is fixed-parameter tractable with respect to the number of edges above the size of a maximum vertex-disjoint P_3-packing [11], but para-NP-hard with respect to the number of edges above the size of a maximum modification-disjoint P_3-packing [48].

Cluster Editing is also a hot topic in the field of algorithm engineering. There are many heuristic approaches that are empirically shown to provide high-quality solutions, as well as exact algorithms. Most algorithms for the latter combine branch-and-bound strategies with integer linear programming as well as heavy preprocessing [20, 44]. Concerning heuristics for Cluster Editing we would like to highlight two approaches which also inspired some of the submissions to the heuristic track and whose quality was empirically verified. The first is the Louvain method by Blondel et al. [16] which is a greedy hill climbing algorithm initially used for community detection. It tries to maximize the relative density of edges inside the communities compared to those outside. The second approach is the so-called FORCE heuristic [66] in which one interprets the edges and non-edges between the vertices as forces and tries to find vertex positionings which minimize the overall energy in the system. Later, Wittkop et al. [67] combined the FORCE heuristic with a parameterized exact algorithm to obtain higher stability in the solution quality.

3 Challenge Setup

There were three tracks in which the participants could compete: an exact, a heuristic, and a new kernelization (data reduction) track. For each track the 200 instances were selected by the Program Committee (PC), half of them publicly available before the submission deadline. The instances were sorted by increasing (n, m) in lexicographic order, where n is the number of vertices and m the number of edges of the particular instance.

In the testing phase the instances were evaluated on `optil.io` [65]. For the final evaluation, we tested the instances on Intel(R) Xeon(R) CPU E5-1620 3.60 GHz machines using the Linux 4.15 kernel. Both evaluations used the same time limits: 30 minutes for the exact track, 10 minutes for the heuristic track, and 5 minutes for the kernelization track.

3.1 Track Descriptions

The exact and the heuristic track followed essentially the same rules as in previous iterations of PACE. The kernelization track was newly introduced and aimed at shrinking the input as much as possible within a five-minute time limit and return an "equivalent" instance. We subsequently provide the details for each track.

Exact Track. In the exact track submissions had to compute an optimal cluster editing set within 30 minutes for the given instance. While no proof of optimality of the returned cluster editing set is required, we disqualified submissions that returned a suboptimal cluster editing set for some instance (a cluster editing set of strictly smaller size was either known to the PC in advance or computed by other submissions). The optimality testing was conducted also on other instances than the 200 instances of the exact track, including some instances of the heuristic track.

The ranking in the exact track is determined by the number of solved instances with the overall summed running time as a tie breaker if two submissions solved the same number of instances.

Heuristic Track. In the heuristic track submissions had to provide a cluster editing set within 10 minutes for a given instance.

The ranking computation for the heuristic track is inherited from the previous iterations of PACE: For each instance, we collected the minimum size $s_{\min}$ of any found cluster editing set (computed by any submission) and the size s of the cluster editing set computed by the submission. The instance score is then $100 \cdot s_{\min}/s$. For example, a score of 100 indicates the submission found was one of the best for this instance while a score of 50 (25) indicates that the submission found a cluster editing set two (four) times as large as a best known cluster editing set. Overall, the score for each instance is in the interval $[0, 100]$ where a score of 0 was given if no cluster editing set was returned within 10 minutes. The total score is simply the average of the instance scores over the 200 test instances.

Kernel Track. The new kernel track was introduced to evaluate preprocessing techniques for Cluster Editing. The rules are inspired by the kernelization concept, which is arguably among the practically most relevant tools from of parameterized algorithmics [35]. It is defined as follows for decision problems: A kernelization algorithm is a polynomial-time algorithm that, given an instance (I, k) of a parameterized problem L, returns an instance (I', k') such that:

1. (I, k) is *equivalent* to (I', k'), that is $(I, k) \in L \iff (I', k') \in L$, and
2. $|I'| + k' \leq f(k)$ for some computable function f.

Note that there are two apparent issues when we want to apply this concept in practice or in a programming contest:

(a) For many problems (including Cluster Editing) the standard parameter k (solution size) is not known in advance but is to be determined by the respective solver.

(b) Instead of deciding whether there is a cluster editing set of a certain size, the task is usually to compute an optimal cluster editing set.

Our solution to issue (a) is straightforward: For an input graph G for Cluster Editing one returns a number d and a graph G' such that $\mathrm{opt}(G) = \mathrm{opt}(G') + d$; here $\mathrm{opt}(H)$ denotes the size of an optimal cluster editing set for graph H. Our solution to issue (b) is inspired by works on enumeration kernels [10, 25] and lossy kernels [50]: We added the requirement that any submission must provide a so-called *lifting algorithm* which takes a (not necessarily optimal) cluster editing set S' for the kernel, and returns a cluster editing set S' for the original instance such that $|S| \leq |S'| + d$. Note that the latter condition accommodates the fact that suboptimal decisions in S' (over which the submission has no control) can be rectified in the solution lifting algorithm. Since computing $\mathrm{opt}(G')$ involves the potentially very time-consuming task of solving Cluster Editing, we did not strictly verify $\mathrm{opt}(G) = \mathrm{opt}(G') + d$ but instead used several heuristic checks: For the 190 out of 200 instances for which we knew $\mathrm{opt}(G)$, we verified that $\mathrm{opt}(G) \geq d$ and $\mathrm{opt}(G) \geq |S'| + d$. Additionally, we checked $|S| \leq |S'| + d$ for each instance and that the returned set S is indeed a cluster editing set for G (three submissions failed this last test and were disqualified). By using submissions from the heuristic track, we ensured that S' is either optimal or close to being optimal. In hindsight, we consider these heuristic tests to be quite efficient in detecting submissions violating the requirements.

For each instance a submission gets $p = (|V'| + |E'| + 1)/(d+1)$ points, where $G' = (V', E')$ is the graph returned by the kernelization algorithm. Similar to the heuristic track, the instance score is then $100 \cdot p_{\min}/p$, where $p_{\min}$ is the minimum points by any submission.

3.2 Selection of Instances

The exact and kernel track shared their instances, the heuristic track had its own set of instances. The instances were drawn from various sources which we describe below in more detail. Most data sources provided weighted instances, that is, for each pair of vertices there is a number given representing some sort of (dis-)similarity of (or distance between) the two vertices. From such instances we generated multiple unweighted instances by adding edges wherever the corresponding weight was above a certain threshold. More specifically, we proceeded as follows: First all edge weights were linearly scaled to be within the interval $[0, 1]$. Then, for each $t \in \{0.1, 0.2, \ldots, 0.9\}$ we created an unweighted graph by adding an edge $\{u, v\}$ whenever the weight for the vertex pair (u, v) is larger than t. Varying thresholds resulted in instances with a very wide range of difficulty (e.g. from solvable within 1 minute to not solved within 3 hours, by a standard ILP formulation [41] solved with Gurobi). A repository with scripts that download and convert all data is available at `https://github.com/PACE-challenge/Cluster-Editing-PACE-2021-instances`.

The data can be categorized as follows:

Biology This category contains two datasets: a real-world biological dataset[1] that contains COG protein similarity data [55, 19] consisting of 3964 weighted instances of which we chose the 155 instances with between 100 and 5,000 vertices, and a dataset with one weighted instance taken from the data accompanying the TransClust[2] clustering tool [67].

Data Mining This category includes two datasets from which six weighted instances were created. The first dataset is from the World Color Survey[3]; the data is converted based on the descriptions of Regier et al. [56] and Thiel et al. [63] and we created one weighted instance. The second dataset is the newsgroups dataset from scikit-learn[4] [54]; the data is converted based on the descriptions of Thiel et al. [63] and we created five weighted instances.

SNAP This category includes instances found in the SNAP [47] dataset. We took 35 large unweighted graphs having 4,000 up to 2 million vertices. These instances were only used in the heuristic dataset.

Random We used randomly generated data to produce some challenging instances. In particular we randomly created action sequences (sequences of actions performed by a person during computer assisted tests as done e.g. at PIAAC [52]) and converted them into graphs as described by Ulitzsch et al. [64].

For the exact and kernel track we tested our instances with a standard ILP formulation [41] solved with Gurobi. We set a time limit of 3 hours per instance and took the running time as indicator of the difficulty of the instances. In the end, we picked 140 instances that were solved within the 30 minutes, 15 instances that were solved in more than 30 minutes but less than 3 hours, and 45 instances that could not be solved within 3 hours. This resulted in 79 graphs from the Biology category, 13 graphs from the Data Mining category, and 108 graphs from the Random category.

For the heuristic track we picked the data sets such that we had an even distribution with respect to the graph size. This resulted in 84 graphs from the Biology category, 43 graphs from the Data Mining category, 36 graphs from the SNAP category, and 37 graphs from the Random category. Figure 2 displays the number of vertices and edges in the selected instances of the complete dataset.

[1] The dataset is available at `https://bio.informatik.uni-jena.de/data/#cluster_editing_data`.
[2] The dataset is available at `https://transclust.compbio.sdu.dk/main_page/index.php`.
[3] The dataset is available at `http://www.icsi.berkeley.edu/wcs/data.html`
[4] The dataset is available at `https://scikit-learn.org/0.19/datasets/twenty_newsgroups.html`

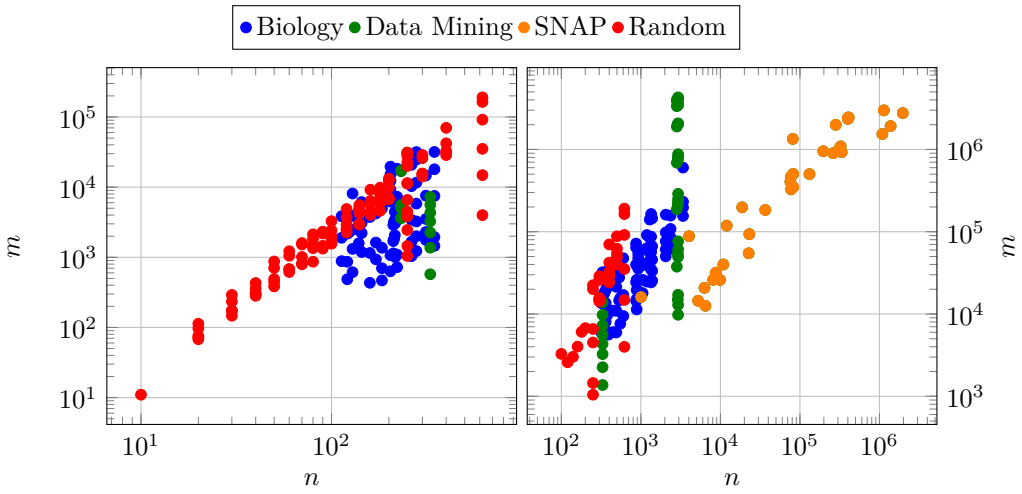


Figure 2 The number of vertices (n) and edges (m) in the two created datasets (left: exact and kernel track; right: heuristic track). In the heuristic track, the first instance with 10 vertices and 31 edges is not shown in order to not clutter the remaining data points too much.

4 Participants and Results

There were 15, 11, and 6 teams that officially submitted a solution to the exact, heuristic, and kernel track, respectively. Several teams participated in more than one track; in total there were 21 distinct teams with 11 of them being student teams (the implementation is done solely by bachelor / master / PhD students). There were roughly twice as many users that submitted a solution to the `optil.io` server during the testing phase. For each track, the top five on `optil.io` were from participants of PACE 2021. The participants represented three continents and the following 11 countries (number of authors from the respective country is given in brackets): Germany (39), Czechia (6), France (5), Australia (4), India (4), United States (3), Japan (2), Mexico (1), Netherlands (1), Poland (1), and the United Kingdom (1). The results are listed below.

4.1 Exact Track

The ranking for the exact track is listed subsequently; see Figure 3 for an illustration of the performance of the accepted solvers on the full benchmark instances. We list the number of solved instances from the 100 hidden instances and in brackets from the 200 overall instances.

1. Lars Gottesbüren, Tobias Heuer, Thomas Bläsius, Philipp Fischbeck, Michael Hamann, Jonas Spinner, Christopher Weyand, Marcus Wilhelm (Karlsruhe Institute of Technology, Hasso Plattner Institut) solved **87** (171) instances [38].
   `https://github.com/kittobi1992/cluster_editing`
2. Alexander Bille, Dominik Brandenstein, Emanuel Herrendorf (Philipps University of Marburg) solved **81** (160) instances [12].
   `https://github.com/EmanuelHerrendorf/pace-2021`
3. Valentin Bartier, Gabriel Bathie, Nicolas Bousquet, Marc Heinrich, Théo Pierron, Ulysse Prieto (Grenoble INP, École Normale Supérieure de Lyon, Université de Lyon, University of Leeds) solved **77** (156) instances [7].
   `https://github.com/valbart/pace-2021`

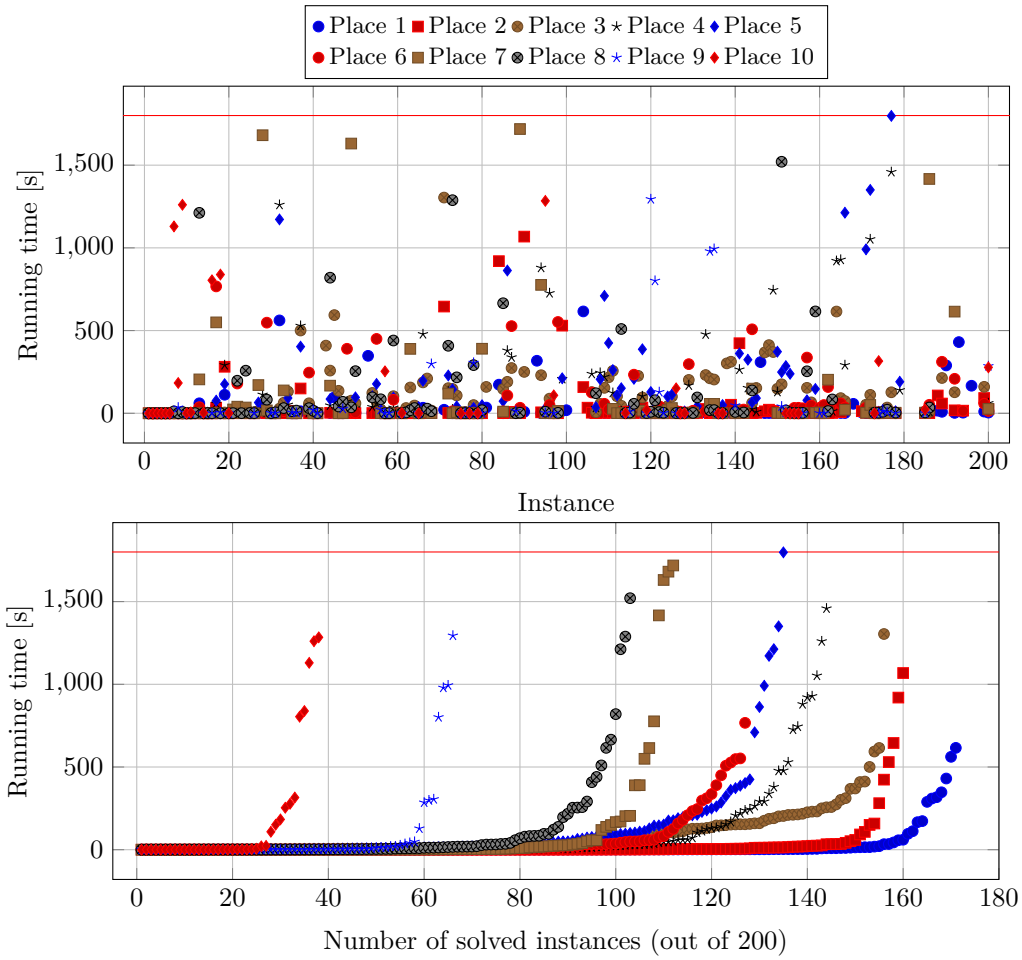


Figure 3 Performance of the top 10 solvers in the exact track. Top: running time plotted for each of the 200 benchmark instances. Bottom: a cactus plot, here a data point with coordinates (x, y) indicates that the corresponding solver could solve x instances of the benchmark set in y seconds per instance. Note that if two solvers solve the same amount of instances within a given time, then the actual set of solved instances can be different (see the top plot). The red horizontal line indicates the timeout of 30 minutes.

4. Jona Dirks, Mario Grobler, Tobias Meis, Roman Rabinovich, Yannik Schnaubelt, Sebastian Siebertz, Maximilian Sonneborn (University of Bremen, Technische Universität Berlin) solved **71** (144) instances [31].
   `https://gitlab.informatik.uni-bremen.de/parametrisierte-algorithmen/java/pace-2021-paca-java`
5. Thorben Freese, Jakob Gahde, Mario Grobler, Roman Rabinovich, Fynn Sczuka, Sebastian Siebertz (University of Bremen, Technische Universität Berlin) solved **67** (135) instances [36].
   `https://gitlab.informatik.uni-bremen.de/parametrisierte-algorithmen/python/paca-python`
6. Yosuke Mizutani (University of Utah) solved **63** (127) instances [51].
   `https://github.com/mogproject/cluster-editing-2021`
7. Václav Blažej, Radovan Červený, Dušan Knop, Jan Pokorný, Šimon Schierreich, Ondřej Suchý (Czech Technical University in Prague) solved **59** (112) instances [13].
   `https://gitlab.fit.cvut.cz/pace-challenge/2021/goat/exact`

8. Sachin Agarwal, Sahil Bajaj, Ojasv Singh, Srinibas Swain (IIIT Guwahati) solved **52** (103) instances [2].
   `https://github.com/sachin-4099/PACE_2021_Cluster_Editing`
9. Sebastian Paarmann (Technische Universität Hamburg) solved **36** (66) instances [53].
   `https://github.com/spaarmann/cluster-editing`
10. Tomoki Takayama (Osaka Prefecture University) solved **17** (38) instances [62].
    `https://github.com/workhouse-lab/pace-2021`

- Sylwester Swat (Poznań University of Technology) solved *all* **100** (200) instances but gave suboptimal cluster editing sets on additional test data [61].
  `https://github.com/swacisko/pace-2021`
- Mario Grobler, Roman Rabinovich, Sebastian Siebertz (University of Bremen, Technische Universität Berlin) solved **95** (190) instances but gave suboptimal cluster editing sets on additional test data [40].
  `https://gitlab.informatik.uni-bremen.de/parametrisierte-algorithmen/cc/pace-2021-paca-cpp`
- Moritz Lichter, Oliver Bachtler, Tim Bergner, Irene Heinrich, Alexander Schiewe (TU Darmstadt, TU Kaiserslautern) solved **71** (142) instances but had errors on 5 further instances [49].
  `https://gitlab.rlp.net/aschiewe/alphabetic`
- Kenneth Dietrich, Mario Grobler, Ozan Heydt, Roman Rabinovich, Sebastian Siebertz, Nick Siering, Leon Stichternath, Julian Tat (University of Bremen, Technische Universität Berlin) solved **46** instances but provided suboptimal cluster editing sets on 37 further instances [30].
  `https://gitlab.informatik.uni-bremen.de/parametrisierte-algorithmen/rust/ceperus/-/tree/v1.0.0`

Strategies Used in the Submissions

At the heart of all submissions we find a branch-and-bound algorithm, an ILP solver, or a combination of the two.

All but two submissions (5th and 8th place) use a branch-and-bound approach. At the core of these algorithms is a search tree algorithm that resolves all induced P_3's: this could be a trivial search tree [22], an improved search tree with more case distinctions [39], or the merge branching strategy which is at the core of the theoretically fastest search-tree algorithm [17]. Only Bartier et al. (3rd place) use, to the best of our knowledge, a new branching which starts with each vertex in its own cluster and then merges and reorders clusters; see their solver description for more details. The other submissions (including places 1, 2, and 4 from the top 5) use one of the existing search trees. Even the best theoretical bound on the search-tree size of $O(1.62^k)$ [17] is prohibitively large for e. g. $k \geq 100$ (which is the case in 180 of the 200 instances). Hence, the "bound"-part in the branch-and-bound approach is crucial.

Most submissions employ data reduction rules as well as lower and upper bounds to prune the search tree. There exist various data reduction rules [11, 20, 23, 24, 26, 33, 39, 42], many of which were implemented in several submissions. Interestingly, Gottesbüren et al. (1st place) described new data reduction rules that are apparently very effective; see their solver description for more details. The lower bounds are based on packing disjoint subgraphs. The easiest candidate (included in almost all submissions) is to compute a set $\mathcal{P}$

of modification-disjoint induced P_3's (that is, two P_3's in the packing share at most one vertex) that is as large as possible: any cluster editing set for the instance has size at least $|\mathcal{P}|$ as at least one edge needs to be modified in each P_3 in $\mathcal{P}$. An improvement of this idea is to find packing of subgraphs where more than one edge modification is needed. As an example, Gottesbüren et al. (1st place) and Bartier et al. (3rd place) looked to also include stars in their packing as in each induced $K_{1,\ell}$ at least $\ell - 1$ edges need to be modified. The last type of employed lower bounds is based on the LP-relaxation of the standard ILP formulation, as done by Dirks et al. (4th place). The upper bounds are mostly described in the heuristic track.

The ILP-based approaches work with the standard ILP formulation [41] that has a variable for each possible edge (each vertex pair) and a constraint for each triple of vertices to ensure the resulting graph is P_3-free. Agarwal et al. (8th place) solved the ILP-formulation with the open source solver CBC. Other submissions combined the ILP-solver with initial data reduction, row generation techniques, and the branch-and-bound solver by first measuring some graph parameters and then decide whether to branch or to use the ILP. These approaches were pursued by Bartier et al. (3rd), Dirks et al. (4th), and Freese et al. (5th).

4.2 Heuristic Track

The ranking for the heuristic track based on the 100 hidden instances is as follows (see Figure 4 for an illustration of the performance of the solvers on all 200 benchmark instances):

1. Lars Gottesbüren, Tobias Heuer, Thomas Bläsius, Philipp Fischbeck, Michael Hamann, Jonas Spinner, Christopher Weyand, Marcus Wilhelm (Karlsruhe Institute of Technology, Hasso Plattner Institut) got an average score of **99.9989**/100 [38].
   `https://github.com/kittobi1992/cluster_editing`
2. Sylwester Swat (Poznań University of Technology) got an average score of **99.9985**/100 [61].
   `https://github.com/swacisko/pace-2021`
3. Valentin Bartier, Gabriel Bathie, Nicolas Bousquet, Marc Heinrich, Théo Pierron, Ulysse Prieto (Grenoble INP, École Normale Supérieure de Lyon, Université de Lyon, University of Leeds) got an average score of **99.9975**/100 [6].
   `https://github.com/GBathie/pace_2021_mu_solver`
4. Martin Josef Geiger (University of the Federal Armed Forces Hamburg) got an average score of **99.9876**/100 [37].
   `https://doi.org/10.5281/zenodo.4891323`
5. Emir Demirović (Delft University of Technology) got an average score of **99.9786**/100 [29].
   `https://bitbucket.org/EmirD/pace-2021/`
6. Ben Strasser got an average score of **99.9723**/100 [60].
   `https://github.com/ben-strasser/cluster-editing-pace2021`
7. Angus Ritossa, Paula Tennent, Tiana Tsang Ung, Akshay Valluru (UNSW Sydney) got an average score of **99.8656**/100 [57].
   `https://bitbucket.org/randomsampling/pace21/`
8. Sachin Agarwal, Sahil Bajaj, Ojasv Singh, Srinibas Swain (IIIT Guwahati) got an average score of **99.6739**/100 [3].
   `https://github.com/sahilbajaj82/PACE-2021-Cluster-Editing`
9. Václav Blažej, Radovan Červený, Dušan Knop, Jan Pokorný, Šimon Schierreich, Ondřej Suchý (Czech Technical University in Prague) got an average score of **99.4946**/100 [14].
   `https://gitlab.fit.cvut.cz/pace-challenge/2021/goat/heuristic`

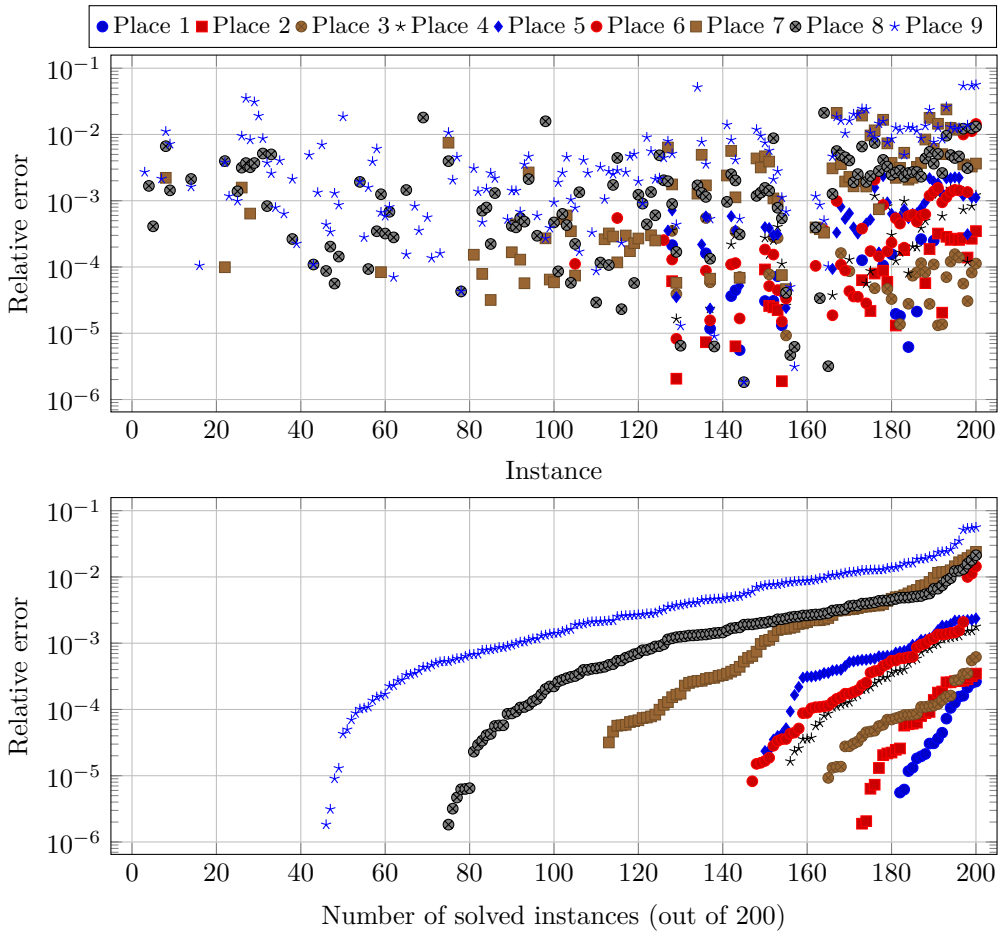


Figure 4 The relative error made by the top nine heuristic submissions (all submissions with an average score higher than 99/100). More precisely, the y-value of a dot is (solution size of submission)/(best known solution size) − 1. In the top plot, the x-axis denotes the respective instance of the benchmark set. In the bottom plot (cactus plot), the x-axis denotes the number of instances where the submission returned a solution with relative error at most the data point's y-value. If a data point is missing (in either plot), then the submission returned a best known solution and the relative error is zero. We remark that the last 20 instances all have solution sizes of more than 300,000 edges (up to 2,500,000 edges). Thus, a relative error of 1% can mean a difference of several thousand edges to the best solution.

10. Jona Dirks, Mario Grobler, Tobias Meis, Roman Rabinovich, Yannik Schnaubelt, Sebastian Siebertz, Maximilian Sonneborn (University of Bremen, Technische Universität Berlin) got an average score of **89.0009**/100 [31]. `https://gitlab.informatik.uni-bremen.de/parametrisierte-algorithmen/java/pace-2021-paca-java`
11. Joshua Harmsen and A.J. Zuckerman (Hamilton College) got an average score of **77.1234**/100 [43].
`https://github.com/joshuaharmsen845/PACE-Challenge/tree/sol1`

Strategies Used in the Submissions

Before going into somewhat more details of solution strategies, we discuss a more efficient representation of solutions employed by most submissions. Instead of maintaining sets of edges, one maintains a partition of the vertices with the meaning that each part in the partition forms a clique in the resulting graph. We will refer to the parts in the partition as *clusters*. It is straightforward to translate solutions between these two representations.

All submissions in the top ten incorporate some form of local search. The differences in the submissions were in how much focus was given to the local search part: Some implementations started with a trivial solution and solely focused on the local search part; herein, by trivial solution we mean a solution in which each vertex is in its own cluster or all vertices are in one cluster. Submissions following this strategy include places 1, 3, 4, 5, and 6. We remark that the 3rd placed submission by Bartier et al. first preprocessed the input using data reduction rules. The only other submission employing data reduction rules is by Sylwester Swat (2nd place).

The submissions of places 2, 7, 8, 9, and 10 all used different heuristics to compute the initial solution. Notably, among these submissions, the 2nd placed submission does have the most elaborate local search part. Thus, local search seems overall the most promising heuristic approach to cluster editing and we subsequently describe different options for what local changes were considered by the participants and what were strategies to avoid getting stuck in local optima. The most frequently used local operations are:

1. (The easiest and by far most-frequently used operation.) Moving a vertex v from one cluster C into another cluster C'. Some submissions only consider moving v to clusters C' that contain neighbors of v as these are the only options that could improve the current solution.
2. Putting a vertex v into a newly created cluster; the new cluster then only contains v.
3. Merging two clusters C and C' into one new cluster.
4. Swapping two vertices, that is, removing two vertices from their cluster and adding them to the respective other cluster.

We remark that the list is not exhaustive and variations of the above operations have been employed as well. To avoid getting stuck in local optima, several strategies have been used, that fall broadly on the following two approaches:

1. Restart the computation from scratch. Here the local operations use randomization so it is unlikely to get stuck in the same local optimum.
2. Perform some local changes that do not improve the solution, that is, the cost of the new solution is at least as high as the old solution. These changes could randomly reassign a fixed number or a fraction of vertices to new clusters or temporarily change the cost function for a fixed number of rounds (e.g. temporarily making edge insertions twice as expensive as edge deletions).

In both approaches, the best encountered solution is stored and returned at the end of the program. Notably, the 1st and 3rd place submissions follow the second approach and do not restart computations from scratch.

4.3 Kernel Track

The ranking for the kernel track is as follows:

1. Sylwester Swat (Poznań University of Technology) got an average score of **65.6761**/100 [61].
   `https://github.com/swacisko/pace-2021`

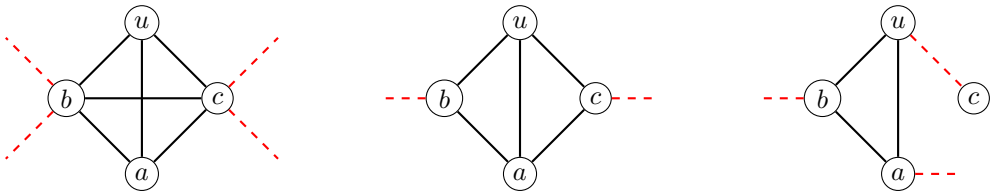


Figure 5 A visualization of the three cases in Reduction Rule 1. The red dashed edges are all present in the input graph and will be removed by the data reduction rule.

- Valentin Bartier, Gabriel Bathie, Nicolas Bousquet, Marc Heinrich, Théo Pierron, Ulysse Prieto (Grenoble INP, École Normale Supérieure de Lyon, Université de Lyon, University of Leeds) got an average score of **71.0077**/100 but their lifting algorithm did not provide valid cluster editing sets on 9 instances [5].
  `https://framagit.org/theo_pierron/pace-2021`
- Václav Blažej, Radovan Červený, Dušan Knop, Jan Pokorný, Šimon Schierreich, Ondřej Suchý (Czech Technical University in Prague) got an average score of **54.0123**/100 but did not provide a lifting algorithm in time (the submission after the deadline passed all our tests) [15].
  `https://gitlab.fit.cvut.cz/pace-challenge/2021/goat/kernelization`
- Moritz Beck, Timon Behr, Johannes Blum, Sabine Cornelsen, Sabine Storandt (University of Konstanz) got an average score of **31.0164**/100 but their lifting algorithm did not provide valid cluster editing sets on 61 instances [8].
  `https://bitbucket.org/moritzbeck/supercereal/`
- Kenneth Dietrich, Mario Grobler, Ozan Heydt, Roman Rabinovich, Sebastian Siebertz, Nick Siering, Leon Stichternath, Julian Tat (University of Bremen, Technische Universität Berlin) got an average score of **26.0103**/100 but their lifting algorithm did not provide a valid cluster editing set on 1 instance [30].
  `https://gitlab.informatik.uni-bremen.de/parametrisierte-algorithmen/rust/ceperus/-/tree/v2.0.0`
- Jona Dirks, Mario Grobler, Tobias Meis, Roman Rabinovich, Yannik Schnaubelt, Sebastian Siebertz, Maximilian Sonneborn (University of Bremen, Technische Universität Berlin) got an average score of **18.0**/100 but did not provide a lifting algorithm [31].
  `https://gitlab.informatik.uni-bremen.de/parametrisierte-algorithmen/java/pace-2021-paca-java`

Strategies Used in the Submissions

We briefly discuss the data reduction techniques used in the submissions to the kernel track. Note that many submissions from the exact and also some from the heuristic track also employ a subset of these techniques. Various submissions employ (subsets of) existing data reduction rules [11, 20, 23, 24, 26, 33, 39, 42]. While refraining from listing these established rules, let us mention two new rules employed in the submissions. Bartier et al. (kernel track) provided the following rule that deals with low degree vertices occurring in a triangle (see Figure 5 for an illustration).

▶ **Reduction Rule 1** (Bartier et al.)**.** Let u be a vertex with neighborhood $\{a, b, c\}$ and let $L = \{a, b, c, u\}$.

- If the vertices in L induce a K_4, a has degree three, and b and c both have degree at most 5, then isolate L. Here, isolating L means removing all edges with exactly one

endpoint in L and reducing k accordingly.
- If the vertices in L induce a diamond (a K_4 minus one edge) and a, b, and c have all degree at most three, then isolate L.
- If $G[\{a, b, c\}]$ contains only the edge $\{a, b\}$ and a and b have all degree at most three, then isolate $\{a, b, u\}$.

Gottesbüren et al. (1st place in exact track) also provided some additional data reduction rules. Among these, the following rule using lower and upper bounds, while simple, proved particularly effective.

▶ **Reduction Rule 2** (Gottesbüren et al.)**.** If modifying an edge e would raise the lower bound above the current upper bound, then e is not allowed to be modified.

Of course Reduction Rule 2 highly depends on the used upper and lower bounds. However, it would be interesting to see whether this or a similar rule could be used to show a problem kernel with respect to some above-guarantee parameterization (recall that the number k of edge modifications is rather large on the benchmark instances).

5 PACE Organization

The Program Committee of PACE 2021 consisted of André Nichterlein (chair), Leon Kellerhals, Tomohiro Koana, and Philipp Zschoche, all from Technische Universität Berlin. During the organization of PACE 2021 the Steering Committee (SC) was composed of

Édouard Bonnet	LIP, ENS Lyon,
Holger Dell	Goethe University Frankfurt and IT University of Copenhagen,
Johannes Fichte	Technische Universität Dresden,
Markus Hecher	Technische Universität Wien,
Bart M. P. Jansen (chair)	Eindhoven University of Technology,
Łukasz Kowalik	University of Warsaw,
Marcin Pilipczuk	University of Warsaw, and
Manuel Sorge	Technische Universität Wien.

In July 2021, André Nichterlein joined the SC, while Édouard Bonnet left. The Program Committee of PACE 2022 will be chaired by Christian Schulz (University of Heidelberg).

6 Conclusion

We thank all participants for their enthusiasm and impressive work and look forward to PACE 2022. We hope that future iterations will again feature a kernel track to further push the development of data reduction rules and kernelization algorithms.

We welcome anyone who is interested to add their name to the mailing list on the website `https://pacechallenge.org/` to receive PACE updates and join the discussion. For frequent updates, especially for updates on plans for PACE 2022, also see the `@pace_challenge` Twitter account.

References

1 Networks project, 2017. URL: `http://www.thenetworkcenter.nl`.
2 Sachin Agarwal, Sahil Bajaj, Ojasv Singh, and Srinibas Swain. Cluster editing using ILP (CLIP): An exact solver for cluster editing, 2021. URL: `https://github.com/sachin-4099/PACE_2021_Cluster_Editing`.

3 Sachin Agarwal, Sahil Bajaj, Ojasv Singh, and Srinibas Swain. Conflict reduced best-fit cluster (CoRBeC): A heuristic solver for cluster editing, 2021. URL: https://github.com/sahilbajaj82/PACE-2021-Cluster-Editing.

4 Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine Learning*, 56:89–113, 2004. doi:10.1023/B:MACH.0000033116.57574.95.

5 Valentin Bartier, Gabriel Bathie, Nicolas Bousquet, Marc Heinrich, Théo Pierron, and Ulysse Prieto. PACE 2021 kernelization track, 2021. doi:10.5281/zenodo.4947841.

6 Valentin Bartier, Gabriel Bathie, Nicolas Bousquet, Marc Heinrich, Théo Pierron, and Ulysse Prieto. Pace 2021 muSolver, 2021. doi:10.5281/zenodo.4947325.

7 Valentin Bartier, Gabriel Bathie, Nicolas Bousquet, Marc Heinrich, Théo Pierron, and Ulysse Prieto. valbart/pace-2021: PACE 2021 exact track, 2021. doi:10.5281/zenodo.4935569.

8 Moritz Beck, Timon Behr, Johannes Blum, Sabine Cornelsen, and Sabine Storandt. Super Cereal, 2021. doi:10.5281/zenodo.4892806.

9 Amir Ben-Dor, Ron Shamir, and Zohar Yakhini. Clustering gene expression patterns. *Journal of Computational Biology*, 6(3-4):281–297, 1999. doi:10.1089/106652799318274.

10 Matthias Bentert, Till Fluschnik, André Nichterlein, and Rolf Niedermeier. Parameterized aspects of triangle enumeration. *Journal of Computer and System Sciences*, 103:61–77, 2019. doi:10.1016/j.jcss.2019.02.004.

11 René van Bevern, Vincent Froese, and Christian Komusiewicz. Parameterizing edge modification problems above lower bounds. *Theory of Computing Systems*, 62(3):739–770, 2018. doi:10.1007/s00224-016-9746-5.

12 Alexander Bille, Dominik Brandenstein, and Emanuel Herrendorf. PACE 2021 exact track submission, 2021. doi:10.5281/zenodo.4889012.

13 Václav Blažej, Radovan Červený, Dušan Knop, Jan Pokorný, Šimon Schierreich, and Ondřej Suchý. GOAT, 2021. URL: https://gitlab.fit.cvut.cz/pace-challenge/2021/goat/exact.

14 Václav Blažej, Radovan Červený, Dušan Knop, Jan Pokorný, Šimon Schierreich, and Ondřej Suchý. GOAT, 2021. URL: https://gitlab.fit.cvut.cz/pace-challenge/2021/goat/heuristic.

15 Václav Blažej, Radovan Červený, Dušan Knop, Jan Pokorný, Šimon Schierreich, and Ondřej Suchý. GOAT, 2021. URL: https://gitlab.fit.cvut.cz/pace-challenge/2021/goat/kernelization.

16 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*, 10, 2008. doi:10.1088/1742-5468/2008/10/P10008.

17 Sebastian Böcker. A golden ratio parameterized algorithm for cluster editing. *Journal of Discrete Algorithms*, 16:79–89, 2012. doi:10.1016/j.jda.2012.04.005.

18 Sebastian Böcker and Jan Baumbach. Cluster editing. In *Proceedings of the 9th Conference on Computability in Europe (CiE 2013)*, volume 7921 of *LNCS*, pages 33–44. Springer, 2013. doi:10.1007/978-3-642-39053-1_5.

19 Sebastian Böcker, Sebastian Briesemeister, Quang Bao Anh Bui, and Anke Truß. Going weighted: Parameterized algorithms for cluster editing. *Theoretical Computer Science*, 410(52):5467–5480, 2009. doi:10.1016/j.tcs.2009.05.006.

20 Sebastian Böcker, Sebastian Briesemeister, and Gunnar W. Klau. Exact algorithms for cluster editing: Evaluation and experiments. *Algorithmica*, 60(2):316–334, 2011. doi:10.1007/s00453-009-9339-7.

21 Édouard Bonnet and Florian Sikora. The PACE 2018 parameterized algorithms and computational experiments challenge: The third iteration. In *Proceedings of the 13th International Symposium on Parameterized and Exact Computation (IPEC '18)*, volume 115 of *LIPIcs*, pages 26:1–26:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.IPEC.2018.26.

22 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.

23 Yixin Cao and Jianer Chen. Cluster Editing: Kernelization based on edge cuts. *Algorithmica*, 64(1):152–169, 2012.

24 Jianer Chen and Jie Meng. A $2k$ kernel for the cluster editing problem. *Journal of Computer and System Sciences*, 78(1):211–220, 2012.

25 Nadia Creignou, Arne Meier, Julian-Steffen Müller, Johannes Schmidt, and Heribert Vollmer. Paradigms for parameterized enumeration. *Theory of Computing Systems*, 60(4):737–758, 2017. doi:10.1007/s00224-016-9702-4.

26 Lucas de O. Bastos, Luiz Satoru Ochi, Fábio Protti, Anand Subramanian, Ivan César Martins, and Rian Gabriel S. Pinheiro. Efficient algorithms for cluster editing. *Journal of Combinatorial Optimization*, 31(1):347–371, 2016. doi:10.1007/s10878-014-9756-7.

27 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 *Proceedings of the 11th International Symposium on Parameterized and Exact Computation (IPEC '16)*, volume 63 of *LIPIcs*, pages 30:1–30:9. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.IPEC.2016.30.

28 Holger Dell, Christian Komusiewicz, Nimrod Talmon, and Mathias Weller. The PACE 2017 parameterized algorithms and computational experiments challenge: The second iteration. In *Proceedings of the 12th International Symposium on Parameterized and Exact Computation (IPEC '17)*, volume 89 of *LIPIcs*, pages 30:1–30:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.IPEC.2017.30.

29 Emir Demirović. Kanpai: A bottom-up approach for cluster editing (PACE 2021), 2021. URL: https://bitbucket.org/EmirD/pace-2021/.

30 Kenneth Dietrich, Mario Grobler, Ozan Heydt, Roman Rabinovich, Sebastian Siebertz, Nick Siering, Leon Stichternath, and Julian Tat. PACA-RUST, 2021. doi:10.5281/zenodo.4884693.

31 Jona Dirks, Mario Grobler, Tobias Meis, Roman Rabinovich, Yannik Schnaubelt, Sebastian Siebertz, and Maximilian Sonneborn. PACA-JAVA, 2021. doi:10.5281/zenodo.4884681.

32 M. Ayaz Dzulfikar, Johannes Klaus Fichte, and Markus Hecher. The PACE 2019 parameterized algorithms and computational experiments challenge: The fourth iteration (invited paper). In *Proceedings of the 14th International Symposium on Parameterized and Exact Computation (IPEC '19)*, volume 148 of *LIPIcs*, pages 25:1–25:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.IPEC.2019.25.

33 Michael R. Fellows, Michael A. Langston, Frances A. Rosamond, and Peter Shaw. Efficient parameterized preprocessing for Cluster Editing. In *Proceedings of the 16th International Symposium on Fundamentals of Computation Theory (FCT '07)*, volume 4639 of *LNCS*, pages 312–321. Springer, 2007. doi:10.1007/978-3-540-74240-1_27.

34 Fedor V. Fomin, Stefan Kratsch, Marcin Pilipczuk, Michał Pilipczuk, and Yngve Villanger. Tight bounds for parameterized complexity of Cluster Editing with a small number of clusters. *Journal of Computer and System Sciences*, 80(7):1430–1447, 2014.

35 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019. doi:10.1017/9781107415157.

36 Thorben Freese, Jakob Gahde, Mario Grobler, Roman Rabinovich, Fynn Sczuka, and Sebastian Siebertz. PACA-PYTHON, 2021. doi:10.5281/zenodo.4884234.

37 Martin Josef Geiger. Source code for PACE 2021, 2021. doi:10.5281/zenodo.4891323.

38 Lars Gottesbüren, Tobias Heuer, Thomas Bläsius, Philipp Fischbeck, Michael Hamann, Jonas Spinner, Christopher Weyand, and Marcus Wilhelm. KaPoCE - an exact and heuristic solver for the cluster editing problem, 2021. doi:10.5281/zenodo.4892524.

39 Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Graph-modeled data clustering: Exact algorithms for clique generation. *Theory of Computing Systems*, 38(4):373–392, 2005.

40 Mario Grobler, Roman Rabinovich, and Sebastian Siebertz. PACE 2021 - cluster editing, c++, 2021. doi:10.5281/zenodo.4889505.

41 Martin Grötschel and Yoshiko Wakabayashi. A cutting plane algorithm for a clustering problem. *Mathematical Programming*, 45(1-3):59–96, 1989. doi:10.1007/BF01589097.

42 Jiong Guo. A more effective linear kernelization for cluster editing. *Theoretical Computer Science*, 410(8-10):718–726, 2009. doi:10.1016/j.tcs.2008.10.021.

43 Joshua Harmsen and A.J. Zuckerman. Cluster editing with existing cliques, 2021. URL: https://github.com/joshuaharmsen845/PACE-Challenge/tree/sol1.

44 Sepp Hartung and Holger H. Hoos. Programming by optimisation meets parameterised algorithmics: a case study for cluster editing. In *Proceedings of the 9th International Conference on Learning and Intelligent Optimization, LION 2015*, volume 8994 of *LNCS*, pages 43–58. Springer, 2015. doi:10.1007/978-3-319-19084-6_5.

45 Christian Komusiewicz and Johannes Uhlmann. Cluster editing with locally bounded modifications. *Discrete Applied Mathematics*, 160(15):2259–2270, 2012.

46 Lukasz Kowalik, Marcin Mucha, Wojciech Nadara, Marcin Pilipczuk, Manuel Sorge, and Piotr Wygocki. The PACE 2020 parameterized algorithms and computational experiments challenge: Treedepth. In *Proceedings of the 15th International Symposium on Parameterized and Exact Computation (IPEC '20)*, volume 180 of *LIPIcs*, pages 37:1–37:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.IPEC.2020.37.

47 Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection, June 2014. URL: http://snap.stanford.edu/data/index.html.

48 Shaohua Li, Marcin Pilipczuk, and Manuel Sorge. Cluster editing parameterized above modification-disjoint P_3-packings. In *Proceedings of the 38th International Symposium on Theoretical Aspects of Computer Science (STACS '21)*, volume 187 of *LIPIcs*, pages 49:1–49:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.STACS.2021.49.

49 Moritz Lichter, Oliver Bachtler, Tim Bergner, Irene Heinrich, and Alexander Schiewe. PACE solver description – Alphabetic, 2021. URL: https://gitlab.rlp.net/aschiewe/alphabetic.

50 Daniel Lokshtanov, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. Lossy kernelization. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC '17)*, pages 224–237. ACM, 2017. doi:10.1145/3055399.3055456.

51 Yosuke Mizutani. PACE 2021 - exact, 2021. doi:10.5281/zenodo.4877899.

52 OECD. Technical report of the survey of adult skills (PIAAC). Technical report, Paris, France, 2013. URL: https://www.oecd.org/skills/piaac/_Technical%20Report_17OCT13.pdf.

53 Sebastian Paarmann. Pandora cluster editing solver, 2021. doi:10.5281/zenodo.4964394.

54 F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

55 Sven Rahmann, Tobias Wittkop, Jan Baumbach, Marcel Martin, Anke Truss, and Sebastian Böcker. Exact and heuristic algorithms for weighted cluster editing. In *Proceedings of the 6th Computational Systems Bioinformatics Conference (CSB '07)*, pages 391–401. World Scientific, 2007. doi:10.1142/9781860948732_0040.

56 Terry Regier, Paul Kay, and Naveen Khetarpal. Color naming reflects optimal partitions of color space. *Proceedings of the National Academy of Sciences*, 104(4):1436–1441, 2007. doi:10.1073/pnas.0610341104.

57 Angus Ritossa, Paula Tennent, Tiana Tsang Ung, and Akshay Valluru. PACE challenge 2021 solver description, from the random sampling group of the UNSW GraphAbility VIP project, 2021. doi:10.5281/zenodo.4946084.

58 Satu Elisa Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007. doi:10.1016/j.cosrev.2007.05.001.

59 Ron Shamir, Roded Sharan, and Dekel Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1-2):173–182, 2004. doi:10.1007/3-540-36379-3_33.

60 Ben Strasser. CELMEC: Cluster editing pace 2021, 2021. URL: https://github.com/ben-strasser/cluster-editing-pace2021.

61 Sylwester Swat. CluES - a heuristic solver for the cluster editing problem, 2021. URL: https://github.com/swacisko/pace-2021.

62 Tomoki Takayama. SatoxaSolver: A submission for PACE 2021, 2021. doi:10.5281/zenodo.4941172.

63 Erik Thiel, Morteza Haghir Chehreghani, and Devdatt P. Dubhashi. A non-convex optimization approach to correlation clustering. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI '19), the 31st Innovative Applications of Artificial Intelligence Conference (IAAI '19), and the 9th AAAI Symposium on EducationalAdvances in Artificial Intelligence (EAAI '19)*, pages 5159–5166. AAAI Press, 2019. doi:10.1609/aaai.v33i01.33015159.

64 Esther Ulitzsch, Qiwei He, Vincent Ulitzsch, Hendrik Molter, André Nichterlein, Rolf Niedermeier, and Steffi Pohl. Combining clickstream analyses and graph-modeled data clustering for identifying common response processes. *Psychometrika*, 86(1):190–214, 2021. doi:10.1007/s11336-020-09743-0.

65 Szymon Wasik, Maciej Antczak, Jan Badura, Artur Laskowski, and Tomasz Sternal. Optil.io: Cloud based platform for solving optimization problems using crowdsourcing approach. In *Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing Companion*, CSCW '16 Companion, pages 433–436. ACM, 2016. doi:10.1145/2818052.2869098.

66 Tobias Wittkop, Jan Baumbach, Francisco P. Lobo, and Sven Rahmann. Large scale clustering of protein sequences with force — a layout based heuristic for weighted cluster editing. *BMC Bioinformatics*, 8(396), 2007. doi:10.1186/1471-2105-8-396.

67 Tobias Wittkop, Dorothea Emig, Sita Lange, Sven Rahmann, Mario Albrecht, John H Morris, Sebastian Böcker, Jens Stoye, and Jan Baumbach. Partitioning biological data with transitivity clustering. *Nature Methods*, 7(6):419–420, 2010. doi:10.1038/nmeth0610-419.

PACE Solver Description: The KaPoCE Exact Cluster Editing Algorithm*

Thomas Bläsius
Karlsruhe Institute of Technology, Germany

Philipp Fischbeck
Hasso Plattner Institute, Potsdam, Germany

Lars Gottesbüren
Karlsruhe Institute of Technology, Germany

Michael Hamann
Karlsruhe Institute of Technology, Germany

Tobias Heuer
Karlsruhe Institute of Technology, Germany

Jonas Spinner
Karlsruhe Institute of Technology, Germany

Christopher Weyand
Karlsruhe Institute of Technology, Germany

Marcus Wilhelm
Karlsruhe Institute of Technology, Germany

Abstract

The cluster editing problem is to transform an input graph into a cluster graph by performing a minimum number of edge editing operations. A cluster graph is a graph where each connected component is a clique. An edit operation can be either adding a new edge or removing an existing edge. In this write-up we outline the core techniques used in the exact cluster editing algorithm of the KaPoCE framework (contains also a heuristic solver), submitted to the exact track of the 2021 PACE challenge.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases cluster editing

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.27

Supplementary Material *Software (Source Code)*: `https://doi.org/10.5281/zenodo.4892524`
Software (Source Code): `https://github.com/kittobi1992/cluster_editing`

1 Preliminaries

Let $G = (V, E)$ be a simple, undirected graph. The cluster editing problem asks to transform G into a disjoint union of cliques with the least number of edge edit operations. An edit is the deletion of an existing edge or the insertion of a missing edge. As a graph is a cluster-graph if and only if it does not contain an induced path on three vertices (a P_3), the problem can also be seen as P_3-free editing.

2 Solver Summary

The solver is based on a more general $\mathcal{F}$-free solver developed for quasi-threshold editing, i.e., $\{C_4, P_4\}$-free editing [6]. Its implementation already solves more than half of the public exact instances in the 2021 PACE challenge without modification. The solver uses a branch-and-bound algorithm that branches on the edges of a forbidden subgraph. In our adaptation, branching is simplified following the work of recent FPT algorithms [3, 2] by generalizing to the weighted cluster editing problem where each vertex pair has an associated integral edit cost. To find the lowest number of edits, the solver actually uses the decision variant that asks if it is possible to solve the instance with k edits. The optimization problem is solved

* This is a brief description of one of the highest ranked solvers of PACE Challenge 2021. It has been made public for the benefit of the community and was selected based on the ranking. PACE encourages publication of work building on the ideas presented in this description in peer-reviewed venues.

by calling the decision variant with increasing values of k. The solver includes a multitude of known reduction rules [4].

Initially, we split the input graph into connected components and solve them individually. After applying all possible reduction rules, the solver branches on the most expensive edge. The edge is marked either as permanent or the vertex pair is forbidden from being connected. Future edits are prevented from changing the state of the vertex pair by setting its edit cost to infinity. As outlined in [3], the endpoints of permanent edges are merged to obtain an equivalent weighted instance of smaller size.

Crucial for the performance of a branch-and-bound algorithm are good lower and upper bounds to prune branches of the search tree that cannot lead to better solutions. Moreover, they enable highly effective parameter-dependent reduction rules [4].

3 Upper Bounds

An initial upper bound is obtained with our heuristic solver. Surprisingly, this initial solution is optimal on all public instances we were able to solve.

4 Lower Bounds

We extend the lower bound from P_3 packings used in the $\mathcal{F}$-free solver [6] to packings of arbitrary structures. We found stars to be particularly effective because they circumvent the problem that the lower bound from P_3 packings cannot be higher than $|E|/2$. For a detailed explanation of lower bounds via subgraph packing and related techniques (e.g., local search) we refer to the $\mathcal{F}$-free solver [6].

5 Reduction Rules

The solver uses the reduction rules 1,2,3 and 5 from [4]. Rule 4, which is based on min-cuts, was found to be ineffective. We implemented the unweighted $4k$ kernel based on critical cliques from [7] and the weighted $2k$ kernel from [5] which is the smallest known kernel for the problem. However, the $2k$ kernel is not used in the final solver since it is mostly dominated by the other reductions and cannot always be applied when dealing with zero-cost edges.

Additionally, the following four reduction rules are used.

Distance Three Reduction

Two vertices with distance three or more cannot be in the same cluster in an optimal solution [1]. Therefore, all vertex pairs with distance three or more are initially marked as forbidden. This does not apply to weighted instances however.

Forced Choices Reduction for all Vertex Pairs

The most effective reduction rule we added is based on lower and upper bounds. If setting an edge to forbidden or permanent would raise the lower bound above the current upper bound, then the opposite edit must be performed. In other words, we identify an edge where, if branched on it, one branch would be pruned immediately.

A naive implementation of this rule is too slow as it requires a quadratic number of lower bound (i.e. packing) computations. However all these packings are similar. Given a packing lower bound for the instance, we locally modify the packing for each vertex pair to obtain the required bounds. Because a packing changes only locally, this can be done significantly

faster than computing n^2 packing lower bounds from scratch. Note that this rule dominates previously known parameter-dependent reduction rules from [4].

Forced Choices Reduction for one Vertex Pair

The locally modified packings from the above reduction are smaller than a heuristically found packing with local search. Thus, we additionally identify a constant number of edits that are highly unlikely to be included in the optimal solution and compute lower bounds for all these edits from scratch. Due to the high quality of the heuristic upper bounds, these two packing based reductions alone are able to solve most instances without branching.

Clique-Like Subgraph Reduction

We analyze the clusters found by the heuristic solver. Using an exact solver as a subroutine for a cluster and its neighborhood, we identify some clusters that are present in an optimal solution. Such clusters can be removed from the initial instance.

References

1 Lucas Bastos, Luiz Satoru Ochi, Fábio Protti, Anand Subramanian, Ivan César Martins, and Rian Gabriel S Pinheiro. Efficient algorithms for cluster editing. *Journal of Combinatorial Optimization*, 31(1):347–371, 2016.

2 Sebastian Böcker. A golden ratio parameterized algorithm for cluster editing. *Journal of Discrete Algorithms*, 16:79–89, 2012.

3 Sebastian Böcker, Sebastian Briesemeister, Quang Bao Anh Bui, and Anke Truß. Going weighted: Parameterized algorithms for cluster editing. *Theoretical Computer Science*, 410(52):5467–5480, 2009.

4 Sebastian Böcker, Sebastian Briesemeister, and Gunnar W Klau. Exact algorithms for cluster editing: Evaluation and experiments. *Algorithmica*, 60(2):316–334, 2011.

5 Yixin Cao and Jianer Chen. Cluster editing: Kernelization based on edge cuts. *Algorithmica*, 64(1):152–169, 2012.

6 Lars Gottesbüren, Michael Hamann, Philipp Schoch, Ben Strasser, Dorothea Wagner, and Sven Zühlsdorf. Engineering Exact Quasi-Threshold Editing. In *18th International Symposium on Experimental Algorithms (SEA 2020)*, volume 160, pages 10:1–10:14. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.SEA.2020.10.

7 Jiong Guo. A more effective linear kernelization for cluster editing. *Theoretical Computer Science*, 410(8-10):718–726, 2009.

PACE Solver Description: ADE-Solver*

Alexander Bille
Computer Science, Philipps Universität Marburg, Germany

Dominik Brandenstein
Computer Science, Philipps Universität Marburg, Germany

Emanuel Herrendorf
Computer Science, Philipps Universität Marburg, Germany

Abstract

This document describes our exact solver "ADE" for the unweighted cluster editing problem submitted to the PACE 2021 competition. The solver's core consists of an FPT-algorithm using a branch and bound strategy in conjunction with several data reduction rules.

2012 ACM Subject Classification Theory of computation → Branch-and-bound; Mathematics of computing → Graph algorithms

Keywords and phrases Unweighted Cluster Editing

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.28

Supplementary Material
Software (Source Code Archive): `https://doi.org/10.5281/zenodo.4889012`
Software (Source Code Repository): `https://github.com/EmanuelHerrendorf/pace-2021`
Software (Source Code Archive Version 2): `https://doi.org/10.5281/zenodo.4889825`
Software (Source Code Repository Version 2): `https://github.com/Araxon/pace-2021-v2/`
Software (Source Code Archive Version 3): `https://doi.org/10.5281/zenodo.4889897`
Software (Source Code Repository Version 3): `https://github.com/Araxon/pace-2021-v3/`

Acknowledgements We want to thank Christian Komusiewicz and Frank Sommer for being our mentors.

1 Introduction

The PACE 2021[1] competition features the Cluster Editing problem. The task of Cluster Editing is to find a minimum size set of edges to add or delete such that the resulting graph is a cluster graph, that is, every connected component is a clique.

Our solver is implemented in Java and its source code can be found at `https://doi.org/10.5281/zenodo.4889012` or via GitHub `https://github.com/EmanuelHerrendorf/pace-2021` . It expects the input graph via the standard input and outputs on the standard output one minimum set of edges to modify to obtain a cluster graph. To avoid stack overflow issues the stack size for the JVM should be increased using the `-Xss` parameter. Our solver is an FPT-algorithm based on a branch and bound strategy. The refined search tree for unweighted cluster editing introduced by Gramm et al. [5] is used as the branching strategy. It guarantees a search tree size of $O(2.27^k)$ where k refers to the minimum number of edge modifications needed to turn the input graph into a cluster graph. During branching edges

* This is a brief description of one of the highest ranked solvers of PACE Challenge 2021. It has been made public for the benefit of the community and was selected based on the ranking. PACE encourages publication of work building on the ideas presented in this description in peer-reviewed venues.

[1] `https://pacechallenge.org/2021`

can be marked as permanent and non-edges as forbidden indicating that these edges need not be modified to obtain an optimal solution.

2 Finding an Optimal Solution

In order to find an optimal solution for Cluster Editing one has to find a valid solution and prove the non-existence of another solution with fewer edge modifications. This can be done for each connected component of the input graph independently, described for example by Böcker et al. [2]. We solve the decision variant of Cluster Editing iteratively asking whether a solution can be found with at most k edge modifications. To arrive at an optimal solution as quickly as possible we try to choose a "good" initial value for k before starting our search tree and then go from there depending on the results we get while traversing it.

For setting the initial value of k we compute a lower and upper bound on the initial graph. These are then heuristically weighted (10% lower bound and 90% upper bound) in order to arrive at an initial value between both of them (version 1 and 3 start with the upper bound decremented by one instead). Also we keep the lower and upper bound and the heuristic solution saved for later use. We then begin traversing the search tree.

If we do not find a solution our current cost limit k is too low so we heuristically increase it by three at a time in order to avoid too many full traversals of unsuccessful search trees and then start traversing the search tree all over. If we have reached the point where we previously already have found a valid solution of size $k + 1$ (either the initial upper bound or another solution found during branching), we simply output this solution. Otherwise we update the lower bound to $k + 1$.

If we, however, do find a solution we have to consider two different cases in order to ensure it is indeed an optimal one:

1. The solution's size matches the current lower bound. Then we can immediately output the solution.
2. The solution is at least one larger than the current lower bound. Then there could still be a valid solution with a size smaller than our current solution. We know, however, that the part of the current search tree which we did already traverse cannot contain it. Otherwise we would have found it earlier instead of the current solution. So we simply continue the traversal of the current search tree with k decremented by one.

3 Branching

Before the branching data reduction rules are applied. After the data reduction, the lower bound is computed to check whether the current branch cannot be solved without exceeding the edge modification limit k. The search tree introduced by Gramm et al. [5] is based on branching on P_3s until no P_3 exists. During the branching a data structure containing all P_3s is maintained and updated according to the modified edges. Furthermore we maintain a conflict graph over all P_3s in which an edge represents two P_3s sharing one non-permanent edge or the non-forbidden non-edge. We define an ordering in which the P_3s are selected by the branching with respect to the following properties of a P_3. Primarily the P_3 with the higher number of permanent or forbidden edges is selected. The following properties will be used to break ties (all in descending order):

1. the number of neighbours in the previously mentioned P_3 conflict graph
2. the degree sum of the three vertices of the P_3

3. the sum of the common neighbours of non-permanent edges and non-common neighbours of non-forbidden non-edges
4. the maximum degree of the vertices of the P_3

If a tie still occurs after these properties have been taken into account, a random P_3 is selected.

4 Lower Bound

The lower bound used for restricting the size of the search tree is the size of a maximal conflict-disjoint P_3 packing which has been suggested by Hartung and Hoos [6]. We follow the incremental approach of Gottesbüren et al. [4] where an initial greedy packing is computed and further improved by local search. For the greedy step the P_3s are taken into account using the same ordering as defined for the branching except that the properties are used in ascending order. The local search tries to exchange one P_3 contained in the packing with at least two P_3s not being part of the packing until no such exchanges can be found. Before the lower bound is updated again, one-to-one exchanges are performed to replace P_3s in the packing with P_3s that are smaller in terms of the ordering.

5 Upper Bound

An upper bound is used in combination with the lower bound to select the k value to start the search tree with. The upper bound used for the parameter k is the size of a heuristically computed solution. This heuristic consists of two steps.

First, the graph is split into clusters using a greedy approach. The closed neighbourhood of a vertex with lowest modification cost is made a clique and removed from the graph until the graph is empty. These costs are the sum of the closed neighbourhood's cut weight and deficiency defined by Cao and Chen [3]. Furthermore, we extended this process such that a vertex from the second neighbourhood of vertex v will be inserted into the clique of v as well if they have at least as many common neighbours as non-common neighbours.

In the second step, these clusters are modified using local search. Three operations being variants of the cluster operations described by Bastos et al. [1] are performed exhaustively until the solution size cannot be decreased anymore. The first operation consists of removing a vertex from a cluster to create its own cluster. The second operation tries to move a vertex into another cluster. The last one merges two clusters. To increase the search area the last two operations are also applied if they leave the solution size unchanged and the total number of such applications does not exceed a threshold.

6 Data Reduction Rules

To reduce the size of the search tree, a set of data reduction rules is applied. We use the $(k+1)$-rule introduced by Gramm et al. [5] and improved by Hartung and Hoos [6]. This rule is applied on the whole graph every fourth search tree node and supplemented by a lighter variant local to recent edge edits and used in the other search tree nodes. If the rule was successfully applied on the whole graph, version 3 afterwards tries to improve the lower bound and then additionally applies the light variant. The following reduction rules are applied in every search tree node. We apply both P_3-rules introduced by van Bevern et al. [7]. Two other reduction rules suggested by Gramm et al. [5] are considered. First, the third (non-)edge of a vertex triple can be directly edited or marked as not editable if the other

two (non-)edges are already not editable and this triple otherwise would form a permanent P_3. The second rule deletes an edge of a P_3 if it is only connected to the rest of the graph through its middle node. Finally, we use a rule of Bastos et al. [1] which sets non-edges $\{u, v\}$ to forbidden if u and v do not have common neighbours.

References

1 Lucas Bastos, Luiz Satoru Ochi, Fábio Protti, Anand Subramanian, Ivan César Martins, and Rian Gabriel S Pinheiro. Efficient algorithms for cluster editing. *Journal of Combinatorial Optimization*, 31(1):347–371, 2016.

2 Sebastian Böcker, Sebastian Briesemeister, and Gunnar W Klau. Exact algorithms for cluster editing: Evaluation and experiments. *Algorithmica*, 60(2):316–334, 2011.

3 Yixin Cao and Jianer Chen. Cluster editing: Kernelization based on edge cuts. *Algorithmica*, 64(1):152–169, 2012.

4 Lars Gottesbüren, Michael Hamann, Philipp Schoch, Ben Strasser, Dorothea Wagner, and Sven Zühlsdorf. Engineering Exact Quasi-Threshold Editing. In Simone Faro and Domenico Cantone, editors, *18th International Symposium on Experimental Algorithms (SEA 2020)*, volume 160 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:14, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

5 Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Graph-modeled data clustering: Exact algorithms for clique generation. *Theory of Computing Systems*, 38(4):373–392, 2005.

6 Sepp Hartung and Holger H. Hoos. Programming by optimisation meets parameterised algorithmics: A case study for cluster editing. In *Learning and Intelligent Optimization: 9th International Conference, LION 9, Lille, France, January 12-15, 2015. Revised Selected Papers*, volume 8994, page 43. Springer, 2015.

7 René van Bevern, Vincent Froese, and Christian Komusiewicz. Parameterizing edge modification problems above lower bounds. *Theory of Computing Systems*, 62(3):739–770, 2018.

PACE Solver Description: PaSTEC - PAths, Stars and Twins to Edit Towards Clusters*

Valentin Bartier
G-SCOP, Grenoble INP, Univ. Grenoble-Alpes, Grenoble, France

Gabriel Bathie
École Normale Supérieure de Lyon, France

Nicolas Bousquet
LIRIS, CNRS, Université Claude Bernard Lyon 1, Université de Lyon, France

Marc Heinrich
University of Leeds, UK

Théo Pierron
LIRIS, CNRS, Université Claude Bernard Lyon 1, Université de Lyon, France

Ulysse Prieto
Independent Researcher, Paris, France

Abstract

This document describes our exact Cluster Editing solver, PaSTEC, which got the third place in the 2021 PACE Challenge.

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

Keywords and phrases cluster editing, exact algorithm, star packing, twins

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.29

Supplementary Material *Software (Source Code)*: `https://github.com/valbart/pace-2021`
Software (Source Code): `https://doi.org/10.5281/zenodo.4935569`

Funding This Work Was Supported by ANR Project GrR (ANR-18-CE40-0032).

1 General description of the solver

Our solver consists in a branch and bound (BB) algorithm. It first starts with a preprocessing phase that we describe below in this section. The algorithm then constructs solutions starting from the initial clustering where all the vertices are in their own clusters of size 1. At each branching step, it either merges two clusters or definitively separate them. We then apply rules to enforce pairs of vertices to be either on the same cluster, or to be in separated clusters. We denote these rules as *enforcing rules*. The branching phase and enforcing rules are described with more details in Section 3.

Overview of the preprocessing phase

We first partition the graph into connected components that will be solved independently one after the other. Indeed, it is not hard to check that no two vertices that belong to separate connected components are in the same cluster of any optimal solution (see for instance [2]). We then compute an upper bound with an heuristic (the reader is referred to [1] for the description of the heuristic). We then label some pairs of vertices as "edges"

* This is a brief description of one of the highest ranked solvers of PACE Challenge 2021. It has been made public for the benefit of the community and was selected based on the ranking. PACE encourages publication of work building on the ideas presented in this description in peer-reviewed venues.

if they necessarily belong to any optimal solution or "non edges" if they do not belong to an optimal solution. This labeling phase permits to avoid unnecessary branchings in the branching phase of the algorithm. We describe this labeling phase of the preprocessing in the next section.

2 Description of the labeling phase

Let us begin with some definitions. The *twinness* of two vertices u, v is the size of the symmetric difference of their neighborhoods. Two adjacent vertices u, v are *i-twins* if their twinness is at most i. Note that 0-twins are what is usually called in the literature *true twins*. One can easily prove that two 0-twins are always in the same cluster of an optimal solution.

On the other hand, two false twins are not necessarily in the same cluster of an optimal solution (consider e.g. a P_3). However, we can prove that if two false twins u, v are not in the same cluster, we can move any of them to the cluster of the other without increasing the cost of an optimal solution. So we can assume that all the false twins are in the same cluster and then we can label as edges pairs of false twins. Similarly, we can label as edge any pair of 1-twins.

One can wonder what can happen when i is increasing. If there is a triangle of 2-twins then we can prove that there exists an optimal solution where there are in the same cluster. Unfortunately, it is not enough to force all these edges. Indeed for the butterfly (two triangles sharing one vertex), there are two triangles of 2-twins but we cannot force all the edges of all the triangles of 2-twins to be in an optimal solution since the optimal solution contains two clusters. We can however prove that if there is a K_4 of 2-twins, these vertices are in the same cluster in any optimal solution.

Generalizing to any i becomes harder and harder since the size of the clique is increasing. Instead we use the following fact, easier to prove: for every $i \leq 8$, if a vertex u has at least $4i-1$ i-twins X then $u \cup X$ are in the same cluster in any possible optimal solution. Note that we do not make any assumption on the twinness of pairs in X nor on the existence of the edges in X. Our proof is computer assisted (the program runs in a few minutes for $i = 8$). We conjecture that this statements holds for any i. However, the lower bound on the number of i-twins might not be tight. For instance, it only gives 7 for $i = 2$ but it might be true that 5 is enough.

Let us summarize this labeling phase:

- For every pair of vertices u, v that are either true twins, false twins, or 1-twins, label (u, v) as an edge.
- For every subset of 4 vertices that forms a K_4 of 2-twins, label the pairs of the K_4 as edges.
- For every $i \leq 7$ and every vertex u such that u has a set X of at least $4i - 1$ i-twins, label the pairs of $u \cup X$ as edges.

3 Description of the branching phase

Let G be the input graph. A *cluster graph* is a partition of the vertices of G into disjoint cliques. The *cost* of graph is the number of edge editions that has to be made to transform G into this solution. We start with a solution where each vertex of the input graph G is in its own cluster (and thus has a cost equal to $|E(G)|$). Furthermore, all along the algorithm we maintain a *current graph*, which is the graph obtained from G by performing the edge editions corresponding to the branchings. The main routine on the branching step is the

following: at each step we choose a pair of clusters C_1 and C_2 to branch on and either merge them (we add all the edges between vertices of C_1 and C_2, and these edges will never be removed in the current branch), or definitively separate them (we remove all the edges between vertices of C_1 and C_2, and no edges between vertices of C_1 and C_2 can be added back in the current branch). We denote edges (resp. non-edges) which cannot be removed (resp. added) in the current branch as *fixed*.

After each branching step we try to:

- cut the branch if possible, or
- merge or separate clusters according to enforcing rules.

Both the cutting of branches and the enforcing rules heavily rely on the computation of an upper bound (that we obtain through an heuristic) and the computation of a lower bound, that we describe below. The rule according to which we cut the branch is rather simple: we cut the branch if the lower bound on the current graph plus its cost (number of edges edited so far) is at least equal to the cost of the upped bound.

Computation of the lower bound

To obtain the lower bound, we consider the current graph and compute how many edges (at least) has to be edited to find a solution. One can note that, for each P_3 on the graph, we have to edit at least one edge. So if we can find a collection of P_3 that pairwise intersect on at most one vertex, the cost of any solution is at least the cost of the current solution plus the number of P_3 in the collection. Unfortunately this rule is too weak since we often need to edit more than $|E|/2$ edges in the instances and the lower bound obtained this way is at most $|E|/2$. We can improve this rule by noting that, for every star $K_{1,\ell}$ the number of edits is at least $\ell - 1$. Our algorithm computes greedily a collection of P_3 that it extends to stars, from which we obtain the lower bound.
The collection of stars is then updated after each branching step. Note that we do not recompute the collection of stars from scratch, but rather only modify the stars that contain at least one endpoint of a pair edited during the current branching step. This allows us to always maintain a lower bound that corresponds to the current graph (and not only a lower bound on the initial graph), while only paying for a reasonable computational cost (since we only perform local modifications).

Linear Programming. For small instances (up to 120 vertices) we run a linear programming algorithm to compute a lower bound using fractional stars. We then round the solution to an integral solution whose size often matches the lower bound (which permits to conclude without the branching algorithm).

Enforcing rules

There are a few cases where we can ensure, during the branching phase, that an edge will or will not be in the final solution, without computing twinness of vertices. Suppose that the current graph contains two clusters C_1, C_2 such that the number of (non yet fixed) non-edges between C_1 and C_2 plus the lower bound on the current graph is greater or equal to the upper bound. Then merging C_1 and C_2 cannot lead to a better solution, and thus we can delete all the C_1, C_2 edges and mark all the C_1, C_2 pairs as fixed non-edges. Similarly if the number of (non yet fixed) edges between the two clusters plus the lower bound is greater or equal to the upper bound, then we add all the missing C_1, C_2 edges and mark alll the C_1, C_2 pairs as fixed edges. We have a total of 5 such rules that we apply after each two branchings.

References

1 Valentin Bartier, Gabriel Bathie, Nicolas Bousquet, Marc Heinrich, Théo Pierron, and Ulysse Prieto. Pace solver description: μsolver – heuristic track. In *IPEC'21*, 2021.

2 Yixin Cao and Jianer Chen. Cluster editing: Kernelization based on edge cuts. *Algorithmica*, 64, August 2010. doi:10.1007/s00453-011-9595-1.

PACE Solver Description: PACA-JAVA*

Jona Dirks
University of Bremen, Germany

Mario Grobler
University of Bremen, Germany

Roman Rabinovich
Technische Universität Berlin, Germany

Yannik Schnaubelt
University of Bremen, Germany

Sebastian Siebertz
University of Bremen, Germany

Maximilian Sonneborn
University of Bremen, Germany

Abstract

We describe PACA-JAVA, an algorithm for solving the *cluster editing problem* submitted for the exact track of the Parameterized Algorithms and Computational Experiments challenge (PACE) in 2021. The algorithm solves the cluster editing problem by applying data-reduction rules, performing a layout heuristic, local search, iterative ILP verification, and branch-and-bound. We implemented the algorithm in the scope of a student project at the University of Bremen.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Cluster editing, parameterized complexity, PACE 2021

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.30

Supplementary Material *Software*: `https://doi.org/10.5281/zenodo.4884681`
Software (Gitlab Repository): `https://gitlab.informatik.uni-bremen.de/parametrisierte-algorithmen/java/pace-2021-paca-java`

1 Introduction

An undirected and simple graph G is a *cluster graph* if its components are cliques (complete graphs). In the *cluster editing problem* we are given an undirected graph G and the problem is to find the minimum set of edge edits (additions or deletions) that turn G into a cluster graph. The *Parameterized Algorithms and Computational Experiments 2021*-Challenge (PACE 2021) is devoted to the cluster editing problem [1]. As a student group at the University of Bremen (under the supervision of Mario Grobler, Roman Rabinovich and Sebastian Siebertz) we participated in the PACE challenge in the scope of a Bachelor project. We implemented an exact algorithm for the cluster editing problem in Java. In the following we describe our approach.

* This is a brief description of one of the highest ranked solvers of PACE Challenge 2021. It has been made public for the benefit of the community and was selected based on the ranking. PACE encourages publication of work building on the ideas presented in this description in peer-reviewed venues.

2 Notation

All graphs in this work are undirected, unweighted and have no self-loops. We use standard graph notation. A *leaf* is a vertex of degree one. For a vertex v we write $N(v)$ for the *neighborhood* of v in G. A path on three vertices is denoted by P_3. We write $P_3(G)$ for the set of induced P_3s of G. If an edge shall not be removed by any following procedure or has been added, we call it *permanent*. Analogously, if it shall not be added or has been removed, we call it *forbidden*. We call two P_3s uvw and xyz *almost disjoint* if $|\{u, v, w\} \cap \{x, y, z\}| = 1$. A *clique* is a subgraph whose vertices are pairwise adjacent. A *cluster graph* is a graph in which every connected component forms a clique. Equivalently, a cluster graph is a graph that contains no induced P_3s. In the *cluster editing* problem we are given an undirected graph G and the task is to compute a set of edge modifications (addition or deletion) of minimum size that transforms G into a cluster graph.

3 Basic Solver Layout

For an input graph G we perform the following steps:

- We delete isolated vertices.
- We apply some data reduction rules to reduce the size of the instance. See Section 4.
- We solve each component individually and combine the solutions afterwards. To solve an individual component C, we first calculate $|P_3(C)|$. Based on graph statistics we either use an ILP or a branch-and-bound-algorithm (see Sections 5–7).

4 Data Reduction Rules

- If $N(v) \setminus \{w\} = N(w) \setminus \{v\}$ and $vw \in E(G)$ for $v, w \in V(G)$, then the edge vw is made permanent. The vertices u and v are twins and are guaranteed to lie in one cluster of the solution by the result of [2].
- The Single-Path-Reduction-Rule starts by finding a leaf. Then the outgoing path p gets followed until a vertex v with degree$(v) > 2$ is reached. Afterwards every second edge on this path gets removed. This happens only if $|p| > 1$. It is easy to see that this is a valid data reduction rule.

5 Finding P_3s

We compute the set of P_3s by using the approach described in Spinner' thesis [8]. For every $v \in V(G)$ and all $u, w \in N(u)$ with $u < w$ we check whether $uw \in E(G)$. The triple uvw is a P_3 if and only if $uw \notin E(G)$. To speed up the calculation we used the matrix datastructure from the Jeigen wrapper [6]. To check if a triple is a P_3 instead of checking the graph, we can retrieve the information out of the matrix, which in the end accelerates the P_3 calculation significantly. If the ratio between the number of induced P_3s and edges is at most 0.005 or the graph has at most 8000 edges, we continue to process the instance via an ILP based approach. Otherwise, we go into a branch-and-bound approach.

6 ILP

We compute an ILP instance where we add for every pair $i, j \in V(G)$ in the graph a variable x_{ij} to the ILP. The intended meaning of $x_{ij} = 0$ is that i and j belong to the same cluster (have distance 0). The objective is to minimize $\sum_{ij \in E(G)} x_{ij} + \sum_{ij \notin E(G)} (1 - x_{ij})$.

Additionally, if an edge is permanent or forbidden, we restrict the variable accordingly (by setting it to 1 or 0, respectively). For every P_3 uvw in our graph we add the constraint $x_{uv} + x_{vw} - x_{uw} \geq 0$. We solve the ILP with the open source ILP solver SCIP ([4], [5], [7]). We verify whether the returned solution is a valid solution for cluster editing (observe that our ILP only ensures that all *initially present* P_3s are edited). If this is not the case, we add a constraint for each newly generated P_3 and iterate this process until we found a valid solution. This iterative approach turned out to be much faster than adding constraints for all triples $u, v, w \in V(G)^3$.

7 Branching

In our branch-and-bound approach we first compute an upper u and lower bound ℓ as follows.

7.1 Upper Bound

For the upper bound we start by calculating a two-dimensional layout based on the algorithm proposed by Fruchterman and Reingold [3]. We then calculate the clusters based on a distance parameter σ: we add an arbitrary vertex v that is not yet part of a clique into a clique and then add all vertices w with the property that no edge on the path from v to w is longer (in the layout) than σ to the same clique. We repeat this until all vertices are grouped into a clique. We iterate over multiple possible distances. After each iteration we do post-processing. The post-processing is a local-search that consists of merging cliques and shifting one vertex into another clique if that decreases the cost of the solution. We do this exhaustively. For the shift vertex operation we also add one empty clique, to allow for the creation of a new clique.

7.2 Lower Bound

We calculate lower bounds in two different ways. If the graph is big or we have little time left, the lower bound is the number of disjoint and almost disjoint induced P_3s that we approximate with a greedy heuristic. Our second lower bound is the LP relaxation of our ILP. If by chance the LP solution is integral we verify if this is a valid solution for cluster editing and return it if this is the case.

7.3 Branching

After having calculated the upper bound $k_0 = u$ and lower bound ℓ (if $u = \ell$, then we already found a solution) we try to find a solution for $k_1 = u - 1, k_2 = u - 2, \ldots$ until no solution is found for some k_i. We then return the solution for k_{i-1}.

We branch based on the edges. For each vertex pair uv contained in a P_3 we do the following: If $uv \in E(G)$, we set the status to forbidden, and try to solve the resulting graph recursively. If this does not succeed, we set the status to permanent and recurse again. If $uv \notin E(G)$ we proceed by first setting the edge to permanent and try solving the resulting graph recursively. If this does not succeed, we set the edge to forbidden and recurse again. We start with the edge contained in the most P_3. This leads to the needed edge modifications. If we change the status for an edge we use the so called transitive closure to resolve depending P_3s. Given $u, v, w \in V(G)$, if uv and vw are marked as permanent edges, we also mark uw as a permanent edge, as there needs to be a edge or else we get a P_3. In each recursive step, we call the ILP if the graph is simple enough, otherwise we continue based on the procedure branch-and-bound as mentioned before. Furthermore, we use the lower bound to cut branches.

References

1 Parameterized Algorithms and Computational Experiments. PACE 2021 - cluster editing, 2021. URL: https://pacechallenge.org/2021/cluster-editing/.

2 Gunnar Böcker, Sebastian W.Klau and Sebastian Briesemeister. Exact algorithms for cluster editing: Evaluation and experiments. *Algorithmica*, 60(2):316–334, 2009. doi:10.1007/s00453-009-9339-7.

3 Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, 1991. doi:10.1002/spe.4380211102.

4 Gerald Gamrath, Daniel Anderson, Ksenia Bestuzheva, Wei-Kun Chen, Leon Eifler, Maxime Gasse, Patrick Gemander, Ambros Gleixner, Leona Gottwald, Katrin Halbig, Gregor Hendel, Christopher Hojny, Thorsten Koch, Pierre Le Bodic, Stephen J. Maher, Frederic Matter, Matthias Miltenberger, Erik Mühmer, Benjamin Müller, Marc E. Pfetsch, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Christine Tawfik, Stefan Vigerske, Fabian Wegscheider, Dieter Weninger, and Jakob Witzig. The SCIP Optimization Suite 7.0. Technical report, Optimization Online, March 2020. URL: http://www.optimization-online.org/DB_HTML/2020/03/7705.html.

5 Gerald Gamrath, Daniel Anderson, Ksenia Bestuzheva, Wei-Kun Chen, Leon Eifler, Maxime Gasse, Patrick Gemander, Ambros Gleixner, Leona Gottwald, Katrin Halbig, Gregor Hendel, Christopher Hojny, Thorsten Koch, Pierre Le Bodic, Stephen J. Maher, Frederic Matter, Matthias Miltenberger, Erik Mühmer, Benjamin Müller, Marc E. Pfetsch, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Christine Tawfik, Stefan Vigerske, Fabian Wegscheider, Dieter Weninger, and Jakob Witzig. The SCIP Optimization Suite 7.0. ZIB-Report 20-10, Zuse Institute Berlin, March 2020. URL: http://nbn-resolving.de/urn:nbn:de:0297-zib-78023.

6 Hugh Perkins. Java wrapper for eigen c++ fast matrix library, 2016. URL: https://github.com/hughperkins/jeigen.

7 Laurent Perron and Vincent Furnon. Or-tools. URL: https://developers.google.com/optimization/.

8 Jonas Spinner. Weighted $\mathcal{F}$-free edge editing, 2019. URL: https://i11www.iti.kit.edu/_media/teaching/theses/ba-spinner-19.pdf.

PACE Solver Description: KaPoCE: A Heuristic Cluster Editing Algorithm*

Thomas Bläsius
Karlsruhe Institute of Technology, Germany

Philipp Fischbeck
Hasso Plattner Institute, Potsdam, Germany

Lars Gottesbüren
Karlsruhe Institute of Technology, Germany

Michael Hamann
Karlsruhe Institute of Technology, Germany

Tobias Heuer
Karlsruhe Institute of Technology, Germany

Jonas Spinner
Karlsruhe Institute of Technology, Germany

Christopher Weyand
Karlsruhe Institute of Technology, Germany

Marcus Wilhelm
Karlsruhe Institute of Technology, Germany

Abstract

The cluster editing problem is to transform an input graph into a cluster graph by performing a minimum number of edge editing operations. A cluster graph is a graph where each connected component is a clique. An edit operation can be either adding a new edge or removing an existing edge. In this write-up we outline the core techniques used in the heuristic cluster editing algorithm of the **Ka**rlsruhe and **Po**tsdam **C**luster **E**diting (`KaPoCE`) framework [2] (contains also an exact solver), submitted to the heuristic track of the 2021 PACE challenge.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases cluster editing, local search, variable neighborhood search

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.31

Supplementary Material *Software (Source Code)*: `https://doi.org/10.5281/zenodo.4892524`
Software (Source Code): `https://github.com/kittobi1992/cluster_editing`

1 Preliminaries

Let $G = (V, E)$ be a simple and undirected graph. $I(u) := \{v \mid (u, v) \in E\}$ denotes the set of all adjacent vertices of a vertex $u \in V$. A clustering $\mathcal{C} := \{C_1, \dots, C_l\}$ is a partition of the vertex set V into l non-empty and disjoint sets. A set $C \in \mathcal{C}$ is also called a *cluster*. We call a cluster $C \in \mathcal{C}$ a *singleton cluster*, if $|C| = 1$. *Isolating* a vertex u or moving it into its *own* cluster means to assign it to a new singleton cluster $C = \{u\}$. A vertex u is adjacent to a cluster $C \in \mathcal{C}$, if there exists a vertex $v \in I(u)$ with $v \in C$. We can transform G into a cluster graph where each cluster of a clustering $\mathcal{C}$ forms a clique by adding all edges of $E^+ := \{(u, v) \notin E \mid \exists C \in \mathcal{C} : u, v \in C\}$ to E and removing all edges of $E^- := \{(u, v) \in E \mid \exists C \in \mathcal{C} : u \in C \ \wedge \ v \notin C\}$ from E.

2 Refinement Techniques

Label Propagation. The label propagation algorithm is a local search heuristic that was initially proposed by Raghavan et al. [5] for community detection, as well as Karypis et al. [4] for hypergraph partitioning, and it was already successfully applied to the cluster editing problem [1]. The algorithm works in rounds. In each round, we visit the vertices in some

* This is a brief description of one of the highest ranked solvers of PACE Challenge 2021. It has been made public for the benefit of the community and was selected based on the ranking. PACE encourages publication of work building on the ideas presented in this description in peer-reviewed venues.

order (random, increasing degree, or input) and move a vertex to an adjacent cluster that minimizes the number of required edit operations (ties are broken randomly). In addition to all adjacent clusters, we also consider isolating the vertex from its current cluster. The algorithm proceeds until we reach a predefined number of rounds or none of the vertices changed their cluster during a round. It can be initialized with an already existing solution or can be used to create an initial solution by initially assigning each vertex to its own cluster. For many problems, the algorithm converges within a few rounds. However, this was not the case here. We observed that for larger instances roughly 10% of vertices are moved during a round and most of them did not change the number of edits (these moves are called *zero-gain* moves). This naturally perturbs the solution and enables frequent improvements in later iterations of the algorithm (plateau search).

Clique Removal and Splitting. The next two refinement techniques were initially proposed by Bastos et al. [1] for the cluster editing problem. Both algorithms also work in rounds and in each round, we visit the clusters of the current solution in some order. The *clique removal* heuristic tries to remove a cluster from the solution by moving the vertices to an adjacent cluster that minimizes the number of edits (vertex isolation is also considered). The *clique splitting* technique chooses two non-adjacent vertices of the considered cluster as seed vertices and isolates them. Afterwards, we assign the remaining vertices to the cluster of one of the two seed vertices based on which induces less edit operations. If the applied moves increased our objective function, we revert them.

Vertex Swapping. The vertex swapping algorithm proceeds in a similar fashion as our other refinement algorithms (several rounds and we visit the vertices in some order). The algorithm moves a vertex speculatively into an adjacent cluster (could potentially increase the number of required edits) and then tries to move vertices out of the target cluster. We move a vertex out of the target cluster, if it decreases the number of required edits. If the overall operation improves the solution quality, we apply the moves. Otherwise, we revert them.

3 Variable Neighborhood Search

We have implemented two different algorithms that are based on the variable neighborhood search (VNS) heuristic [3]. The general idea of such algorithms is to start from an initial solution and then *perturb* the current solution (also called *shaking* in the literature) followed by *local search* phase to find a new local optimum. This process can be repeated until a predefined number of rounds or a time limit is reached.

Our first algorithm applies a perturbation operator to a large number of vertices (between 1% and 25% of the vertices) and then runs the refinement techniques described in Section 2. The second algorithm only perturbs a few vertices (between 1 and 5) and then executes the label propagation algorithm only initialized with the mutated vertices and their neighbors. We call the first one *global* and the second *highly-localized VNS*. The rationale behind this is that there are instances that converge very quickly and only specific mutations lead to additional improvements. For such instances, the global algorithm is a good choice, since bad decisions are reverted very quickly and the likelihood of mutating the right vertices is very high. On the other hand, there are instances that converge very slowly, and the perturbation of a large number of vertices can drastically degrade the solution quality such that our refinement techniques are not able to restore the initial solution quality. Thus,

performing highly-localized mutations and refinement becomes an important techniques to escape from local optima.

We run the VNS algorithms in an alternating fashion, in short bursts of 20 seconds each. However, we track the improvement made in the last burst, and prefer running the more promising algorithm next.

Initial Solution. Initially, each vertex is assigned to its own cluster. To compute an initial solution, we first run 100 rounds of our label propagation algorithm, then 20 rounds of our vertex swapping algorithm and last, 10 rounds of the clique removal and splitting technique. We repeat this several times (at most 100 times, but we abort after 10 seconds) and use the best solution found as input for our VNS algorithms.

Global Variable Neighborhood Search. Our global VNS algorithm performs one of the following two operations in each step: It either *intensifies* the current solution by applying the refinement techniques described in Section 2 or applies one of our three *perturbation* operators followed by an intensify operation. The decision which operation we choose next is based on the improvement history of the last 5 runs of the corresponding operation. The more an operator improves the solution quality compared to the other, the more likely it is that we execute it next.

We have implemented three different perturbation operators, which are also described in the work of Bastos et al. [1]. Each operator mutates between 1% and 25% of the vertices, which we choose uniformly and at random. The first operator, which we call *vertex isolator*, isolates a vertex. Our *vertex mover* operator, moves a vertex to a random adjacent cluster. The third operator, which we call *clique splitter*, isolates a vertex from its current cluster and then chooses one additional adjacent vertex from its previous cluster to form a new clique of size two.

In an intensify step, we perform 25 rounds of label propagation, 5 rounds of vertex swapping and 10 rounds of clique removal and splitting each. The vertex swapping and clique removal and splitting techniques are only executed with a probability of 50%. After a perturbation operation, we perform 50 rounds of label propagation, but abort early if the estimated improvement (based on the improvement history of label propagation) is greater than our initial edits before the perturbation operation plus 100. Further, we perform another 50 rounds of label propagation, if we are only 25 edits away from the initial number of edits. We perform the other refinement techniques similar to before.

Highly-Localized Variable Neighborhood Search. In each step, our highly-localized VNS algorithm selects between 1 and 5 vertices uniformly at random that are perturbed and, subsequently performs a highly-localized run of label propagation. For each of the selected vertices, we additionally choose an adjacent neighbor that we also perturb with a probability of 75%. As perturbation operator, we either isolate a vertex from its current cluster or move it to a random adjacent cluster. Afterwards, we perform 5 rounds of label propagation restricted to the perturbed vertices and their neighbors. If the overall operation improved the current solution, we apply all moves and otherwise, we revert them.

References

1 Lucas Bastos, Luiz Satoru Ochi, Fábio Protti, Anand Subramanian, Ivan César Martins, and Rian Gabriel S Pinheiro. Efficient Algorithms for Cluster Editing. *Journal of Combinatorial Optimization*, 31(1):347–371, 2016.

2 Thomas Bläsius, Philipp Fischbeck, Lars Gottesbüren, Michael Hamann, Tobias Heuer, Jonas Spinner, Christopher Weyand, and Marcus Wilhelm. KaPoCE - An Exact and Heuristic Solver for the Cluster Editing Problem. `https://github.com/kittobi1992/cluster_editing/tree/pace-2021`, 2021. `doi:10.5281/zenodo.4892524`.

3 Pierre Hansen and Nenad Mladenović. Variable Neighborhood Search. In *Handbook of Metaheuristics*, pages 145–184. Springer, 2003.

4 George Karypis and Vipin Kumar. Multilevel k-way Hypergraph Partitioning. Technical Report 98-036, University of Minnesota, 1998.

5 Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. Near Linear Time Algorithm to Detect Community Structures in Large-Scale Networks. *Physical Review E*, 76(3):036106, 2007.

PACE Solver Description: CluES – a Heuristic Solver for the Cluster Editing Problem*

Sylwester Swat
Institute of Computing Science, Poznań University of Technology, Poland

Abstract

This article briefly describes the most important algorithms and techniques used in the cluster editing heuristic solver called "CluES", submitted to the 6th Parameterized Algorithms and Computational Experiments Challenge (PACE 2021).

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Cluster editing, heuristic solver, graph algorithms, PACE 2021

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.32

Supplementary Material *Software (Source Code)*: `https://doi.org/10.5281/zenodo.4949728`

1 Problem description

A cluster editing set of a graph $G = (V, E)$ is a set of edges $E_{mod} \subseteq V \times V$ such that each connected component of a graph $G[E \Delta E_{mod}]$ is a clique, where $A \Delta B$ denotes the symmetric difference of sets A and B. The solver briefly described here is a heuristic approach to the cluster editing problem, where the goal is to find, for a given graph, a cluster editing set of the smallest size.

2 Solver description

In this paper we provide a short description of the most important algorithms implemented in solver CluES. Due to many parameters used in the implementation and a vast number of case distinctions that need to be taken into account, this description may not contain full information about the algorithms behavior in every possible situation.

All algorithms implemented in CluES operate on a data structure we called a *partition graph*. For a given graph $G = (V, E)$ and a partition $P = \{C_1, C_2, \ldots, C_p\}$ of V, a partition graph G_P is tuple (V_P, E_P, w_n, w_e), where V_P is the node set, E_P is the edge set and $w_n : V_P \mapsto \mathbb{N}$ and $w_e : E_P \mapsto \mathbb{N}$ are weight functions, defined as below:

$$\begin{aligned} V_P &= \{1, 2, \ldots, p\} \\ E_P &= \{(u, v) \in V_P \times V_P : \exists_{x \in V \cap C_u, y \in V \cap C_v} (x, y) \in E\} \\ w_n(u) &= |\{x \in V : x \in C_u\}| \\ w_e(u, v) &= |\{(x, y) \in E : x \in C_u, y \in C_v\}| \end{aligned}$$

It is worth noting here that by setting $V_P = V$, $E_P = E$, $w_n(u) = 1$, and $w_e(u, v) = 1$ we obtain the partition graph representing the original input graph.

* This is a brief description of one of the highest ranked solvers of PACE Challenge 2021. It has been made public for the benefit of the community and was selected based on the ranking. PACE encourages publication of work building on the ideas presented in this description in peer-reviewed venues.

The main algorithm works in independent iterations (main iterations). In each iteration a separate cluster editing set is found (independently from previous iterations) and the best result found in all main iterations is returned as a final cluster editing set. In main iterations, instead of looking for a set of edge modifications, we look for a partition $\{C_1, C_2, ..., C_p\}$ of set V corresponding to clusters in a graph after edge modifications. Each main iteration consists of the following phases:

1. Quickly create a set of feasible solutions using some fast heuristics.
2. Coarsen the graph. To do this, we create a partition $\{S_1, \ldots, S_t\}$ of V such that all nodes in each S_i occur always in the same cluster in all already found solutions. For each $1 \leq i \leq t$ find a maximal matching M_i of a graph $G[S_i]$. We take $M = \bigcup_{1 \leq i \leq t} M_i$ and "contract" (by creating a corresponding partition graph) all edges in M.
3. Repeat the process in steps $1 - 2$ for the coarsened graph. After the steps are repeated (the graph is coarsened) for the R-th time (R is some small integer, usually 3 or 4), proceed to step 4.
4. Consider the best solution found so far and improve the solution using local search.
5. If possible, uncoarsen the graph "by one level" (uncoarsen pairs of nodes from the most recently found matching in step 2) and go back to step 4.

3 Preprocessing

In main iterations we sometimes use preprocessing to determine certain subsets of nodes that are in the same cluster in some optimal solution. This information is used to create a proper partition graph before starting the first phase of a given main iteration. We use over 15 parameter-independent kernelization rules based on edge cuts and critical cliques. Nine of these rules are based on concepts related to critical cliques and can be found in [3] and [2]. One rule is a "similar neighborhoods" rule (see [1]) applied to an unweighted graph. The rule states that, if there exists an edge $(u, v) \in E$ with $N[u] = N[v] \cup \{w\}$ for some $w \notin N(v)$, then there exists an optimal solution with u and v belonging to the same cluster. Another three rules are specifications of the "almost clique rule" [1] applied for all triangles, K_4's and induced diamonds. The "almost clique rule" states that, if there exists a subset $C \subset V$ such that $G[C]$ is connected and the size of the minimum cut in $G[C]$ is not smaller than $|\{(a, b) \in C \times C : (a, b) \notin E\}| + |\{(a, b) \in E : a \in C, b \notin C\}|$, then there exists an optimal solution where all nodes in C belong to the same cluster. The other preprocessing rules are of our own invention and are based on properties of some intersections of neighborhoods of critical cliques and usually do not guarantee that their application will preserve the result optimality property, but are just a reasonable guess that often enables us to quickly and accurately identify nodes that most often belong to the same cluster in some good solution.

4 Local search techniques

In order to improve a solution, we use a local search technique. In each iteration of the local search, we call several procedures. All of them are based on moving some subsets of nodes from a cluster to which they belong in the current solution to some other cluster (possibly creating a new one). We allow moving subsets only if the total number of edges required to create the cluster graph obtained after the move is not larger than before the move. We consider the following possibilities of a move:

1. Move a single node v from its cluster to some other cluster.
2. Move two nodes belonging to the same edge to some cluster.

3. Move three nodes a, b, c with $b \in N(a)$ and $c \in N(b)$ to some cluster.
4. Interchange clusters of a pair of nodes a and b.
5. Move node a to a cluster containing node b and simultaneously move node b to some other cluster.
6. For a given cluster, create some permutation $(u_0, u_1, \ldots, u_x)$ of nodes belonging to that cluster. For each set $U_i = \{u_0, \ldots, u_i\}$, try to move U_i to some other (single) cluster.
7. For a given cluster, greedily move nodes from this cluster to some other cluster, always selecting the one that minimizes some objective function (e.g. the difference in the total number of edge modifications needed to obtain states before and after the move). This way it is possible to move several nodes from the given cluster to multiple clusters.
8. For a given cluster, greedily move nodes from other clusters to this cluster. This way it is possible to move several nodes from multiple clusters to the given cluster.

Let us note here that these steps work best when applied to a properly created partition graph. Moving a single node of a partition graph G_P corresponds to moving some subset of nodes of the original graph G. Hence it is possible to move large subsets of nodes from G by moving just a single node from G_P.

It is important to mention that the crucial point is the design and implementation of algorithms efficiently realizing described steps (working in a weighted scenario, since they are applied for the partition graph structure). Naively checking the moves described above, except perhaps the first one, could lead to a complexity of $\Omega(N^2)$ that would be prohibitive to perform for larger graphs, let alone calling the algorithms hundreds or thousands times during the local search. By efficiently we mean checking all possible structures (all nodes, edges, P_3's and pairs of nodes) that can be moved in steps 1-5, the guarantee of a feasible worst-case running time and the implementation maximally reducing the running time constant.

As an easy example of an algorithm efficiently realizing described steps, let us consider the following subproblem of step 3: "Does there exist a triangle (a, b, c) in G, such that making (a, b, c) a single, separate cluster would result in a better solution?". Naively checking all triangles in the graph could lead to an algorithm with $\Omega(N^2)$ complexity and that would be too slow, e.g., for a graph with $N = 10^5$ nodes and $M = 5 * 10^5$ edges. Enumeration of all triangles can, however, be done in time $O(E^{\frac{3}{2}})$ with a fairly small constant factor (especially in practice), what gives a more practical approach.

5 Availability

The source code of CluES is available at `https://doi.org/10.5281/zenodo.4949728` (please select the proper version of the solver to access required contest track: heuristic, exact or kernelization).

References

1 Sebastian Böcker, Sebastian Briesemeister, and Gunnar Klau. Exact algorithms for cluster editing: Evaluation and experiments. *Algorithmica*, 60:316–334, January 2008.

2 Jianer Chen and Jie Meng. A 2k kernel for the cluster editing problem. *Journal of Computer and System Sciences*, 78(1):211–220, 2012. JCSS Knowledge Representation and Reasoning.

3 Jiong Guo. A more effective linear kernelization for cluster editing. *Theor. Comput. Sci.*, 410:718–726, March 2009.

PACE Solver Description: μSolver – Heuristic Track*

Valentin Bartier
G-SCOP, Université Grenoble Alpes, France

Gabriel Bathie
École Normale Supérieure de Lyon, France

Nicolas Bousquet
LIRIS, CNRS, Université Claude Bernard Lyon 1, Université de Lyon, France

Marc Heinrich
University of Leeds, UK

Théo Pierron
LIRIS, CNRS, Université Claude Bernard Lyon 1, Université de Lyon, France

Ulysse Prieto
Independent researcher, Paris, France

Abstract

This document describes our heuristic Cluster Editing solver, μSolver, which got the third place in the 2021 PACE Challenge. We present the local search and kernelization techniques for Cluster Editing that are implemented in the solver.

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

Keywords and phrases kernelization, graph editing, clustering, local search

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.33

Supplementary Material *Software (Source Code)*: `https://doi.org/10.5281/zenodo.4947325` [1]

Funding This work was supported by ANR project GrR (ANR-18-CE40-0032).

1 Large neighborhood local search for Cluster Editing

Our solver, μSolver [1], uses various local moves to iteratively improve a solution (that is, a partition of the input into clusters) during the allotted time. The initial solution is a trivial solution with n clusters: all vertices belong to the same cluster and the remaining $n-1$ clusters are empty.

The first local move that we use is the `best_move`: it takes as parameter a vertex v and a solution S, and returns a solution S' in which v has been moved to the cluster c that minimizes the cost of S'. One can notice that c is either the cluster of a neighbor of v or an empty cluster. Therefore, the `best_move` function can be implemented to run in $O(d(v))$ time, where $d(v)$ is the degree of v. This local move is not sufficient to obtain good solutions, as it is very sensitive to local minima.

Our second local move, `bfs_greedy`, aims to avoid these local minima by making larger moves. Instead of moving a single vertex at a time, we select a random subset X of t vertices, isolate them in the graph (by moving them to a cluster of size 0), and then insert them back using `best_move`. In order to maximize the efficiency of this move, we fill X with vertices

* This is a brief description of one of the highest ranked solvers of PACE Challenge 2021. It has been made public for the benefit of the community and was selected based on the ranking. PACE encourages publication of work building on the ideas presented in this description in peer-reviewed venues.

that are close to one another. Namely, we call the function `select_BFS` on a vertex v, which performs a BFS starting from v, and returns the first t vertices that it visits. We then use this move on every vertex of the graph: the `select_BFS` function only returns vertices that were not returned before. Running `bfs_greedy` on every vertex can be done in $O(n+m)$.

Our final algorithm works in passes: a pass runs the `bfs_greedy` move on every vertex of every connected component, and runs passes until it timeouts.

2 Parameter learning

Notice that our `bfs_greedy` heuristic requires a parameter t. Our experiments showed a trade-off: setting $t \simeq 15$ yields good results on sparse instances, while setting $t \simeq 50$ yields good results on dense instances. Instead of settling on a single value for every instance, we opted for a learning algorithm: we specify a set of values $t_1, \dots t_\ell$ with associated weights $w_1, \dots, w_\ell$. At each step, we sample a value t_s according to the weights (w_i) and select a set X of size t_s. If the call to the heuristic with this choice of t yields an improved solution, we add 1 to the value w_s.

3 Overview

We combine our local search with some kernelization rules, that apply mostly on sparse instances (see next section). Moreover, we process each connected component of the graph separately, as it is easy to see that vertices from different connected components cannot be in the same cluster of an optimal cluster editing.

Algorithms 1 and 2 give a global overview of the pseudo code of our solver.

Algorithm 1 Main solver.

Input: G, t, w
$G \leftarrow$ `kernelize`(G)
for each connected component C_i **do**
  $S_i \leftarrow$ `trivial_solution`(C_i)
while timeout is not reached **do**
  for each connected component C_i **do**
    $j \leftarrow$ `sample_weights`$(w_1, \dots, w_\ell)$
    $S' \leftarrow$ `bfs_greedy`$(C_i, S_i, t[j])$
    if `cost`$(S') <$ `cost`(S_i) **then**
      $S_i \leftarrow S'$
      $w[j] \leftarrow w[j] + 1$
return $\mathcal{S} = \bigcup_i S_i$

Algorithm 2 `bfs_greedy` heuristic.

Input: C, S, t
$seen \leftarrow [false] * n$
for each $v \in C$ in a random order **do**
  if $seen[v]$ **then**
    go to the next vertex
  $X \leftarrow$ `select_BFS`$(C, v, seen, t)$
  $S' \leftarrow S$
  for u in X **do**
    $S' \leftarrow$ `isolate`(u, S')
  for u in X **do**
    $S' \leftarrow$ `best_move`(u, S')
return S'

4 Kernelization rules

We present here our kernelization rules (see figure) without their safeness proofs due to space constraints. Our algorithm applies these rules until none of them can be applied.

▶ **Rule 4.1.** *Let u be a vertex with either a 1-neighbor or two adjacent 2-neighbors. If u has another neighbor v such that u and v have no common neighbor, then delete uv.*

This implies that we delete all edges but one between each vertex and its 1-neighbors.

▶ **Rule 4.2.** *Let $uvxw$ be an induced C_4 where v and w have degree 2. Delete uv and wx.*

▶ **Rule 4.3.** *Let u be a 3-vertex with neighbors a, b, c.*

- *If ab, bc and ac are edges, a has degree 3 and b, c both have degree at most 5, delete all edges bx and cx for $x \notin \{u, a, b, c\}$.*
- *If ac and bc are edges, ab is a non-edge, and a, b, c all have degree at most 3, delete all edges ax and bx for $x \notin \{u, c\}$.*
- *If ab is an edge but not ac and bc, and a, b both have degree at most 3, delete uc and all edges ax and bx for $x \notin \{u, a, b\}$.*

The last item is actually a special case of a more general rule that we present below (with $k = 3$). The two other items are actually specification designed to handle the case where the disjointness hypothesis is not met anymore. However, we did not implement it since it does not provide a better kernelization on the public instances.

▶ **Rule 4.4.** *Let K be a clique on k vertices such that each vertex outside of K has at most one neighbor in K. For every $u \in K$, denote by $f(u)$ the number of neighbors of u outside of k. Write $K = \{u_1, \dots, u_k\}$ where $f(u_1) \leqslant \cdots \leqslant f(u_k)$. If, for every $i \in [1, k]$, $\sum_{j=i+1}^{k} f(u_j) \leqslant \binom{k}{2} - \binom{i}{2}$, delete all edges with exactly one endpoint in K.*

As a final remark, note that Rule 4.3 allows to solve the Cluster Editing problem in polynomial time on subcubic graphs. Indeed, applying Rule 4.3 removes all the triangles in subcubic graphs. It then remains to find a maximum matching problem in the kernel.

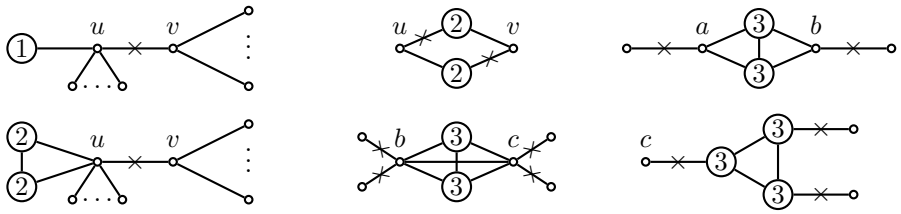


Figure 1 Kernelization rules. Vertices labeled by an integer i are of degree i.

References

1 Valentin Bartier, Gabriel Bathie, Nicolas Bousquet, Marc Heinrich, Théo Pierron, and Ulysse Prieto. Pace 2021 musolver, 2021. doi:10.5281/ZENODO.4947325.

PACE Solver Description: A Simplified Threshold Accepting Approach for the Cluster Editing Problem*

Martin Josef Geiger
Logistics Management Department, University of the Federal Armed Forces Hamburg, Germany

Abstract

We present a simple heuristic for the Cluster Editing Problem as presented in the Parameterized Algorithms and Computational Experiments (PACE) 2021. Our method makes use of a simple Threshold Accepting strategy and employs single neighborhood moves only.

Despite its simplicity, the results of the method are encouraging. However, and this has to be expected, the approach cannot ultimately win in a competitive setting such as PACE 2021. Nevertheless, some interesting insights can be derived from such a simple method, as this gives an idea of how good results can be by a comparable basic approach with a reasonable implementation effort.

2012 ACM Subject Classification Theory of computation → Randomized local search

Keywords and phrases Cluster Editing Problem, Threshold Accepting, Local Search

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.34

Supplementary Material *Software (Source Code)*: `https://doi.org/10.5281/zenodo.4891323`

1 Introduction

In the fall of 2020, the Parameterized Algorithms and Computational Experiments (PACE) competition invited to work on the Cluster Editing Problem. In this problem, an undirected, weighted graph $G = (V, E)$ is to be transformed into a cluster graph by a minimum number of edge modifications (additions and/ or deletions).

Following our contribution to the PACE 2018, and based on our experiences with simple, yet effective local search algorithms in other competitions (with a local search technique reduced to a single-operator search for the VeRoLog Solver Challenge 2019 as a prominent example), we decided to concentrate on what works best with little effort.

2 The actual algorithm

The method implemented for PACE 2021 is best characterized as a multi-start Threshold Accepting-type local search.

2.1 Preprocessing

We did experiment with some preprocessing techniques that allow for a reduction of the graph G. However, in our limited experiments, we did not find them to be overly beneficial, so for the actual submission, all of them have been removed.

* This is a brief description of one of the highest ranked solvers of PACE Challenge 2021. It has been made public for the benefit of the community and was selected based on the ranking. PACE encourages publication of work building on the ideas presented in this description in peer-reviewed venues.

2.2 Creating an initial solution

The initialization of the first alternative is straight-forward: Each node is placed in a separate cluster. Hence, $|V|$ clusters are initially created, and the objective function value of this initial solution is (always) $|E|$.

2.3 Improvements by Local Search

The only move implemented in our contribution is a *single-node-move* operator. This operator investigates the effect of removing a single node from its current clusters, and placing it either

- in a new, i. e., a previously empty cluster, or
- in a "neighboring" cluster.

The first sub-type is important as otherwise the number of clusters would monotonically decrease – and can only decrease – throughout the runs. Note that the definition of a neighboring cluster is based on the elements in E: A node might join another cluster if (and only if) there is an incident edge in E connecting it to a node of this cluster. Effectively, this sub-type can be implemented on the edges as given in G.

Moves are considered in random order.

2.4 Acceptance condition

Any move improving the current solution is accepted right away. Moreover, moves that do not change the current evaluation are accepted with $p = 0.5$. Furthermore, moves that are within a threshold T of the current best-know-solution are accepted with, again, $p = 0.5$. T is, during search, randomly changed every 1,000,000 moves by choosing it from $\{0, 1, 2\}$.

A value of $T = 0$ corresponds to a pure hillclimbing algorithm, while values of $\{1, 2\}$ allow the search to escape local optima. In our experience, and based on the limited work put into this project, this mechanism contributes to the quality of the identified solutions. It has also been observed that accepting moves that do not change the objective function value (but merely maintain it) are beneficial in terms of diversifying the search.

2.5 Restarts

Once the algorithm is unable to improve the current best-known solution for 50 percent of its running time, a restart is triggered. See subsection 2.2 on the initial solution.

The best solution identified through all runs is kept and reported once the termination criterion is reached.

3 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.4891323`.

PACE Solver Description: Cluster Editing Kernelization Using CluES*

Sylwester Swat
Institute of Computing Science, Poznań University of Technology, Poland

Abstract

This article briefly describes the most important algorithms and techniques used in the cluster editing kernelization solver called "CluES", submitted to the 6th Parameterized Algorithms and Computational Experiments Challenge (PACE 2021).

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Cluster editing, kernelization, graph algorithms, PACE 2021

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.35

Supplementary Material *Software (Source Code)*: `https://doi.org/10.5281/zenodo.4949787`

1 Problem description

A cluster editing set of a graph $G = (V, E)$ is a set of edges $E_{mod} \subseteq V \times V$ such that each connected component of a graph $G[E \Delta E_{mod}]$ is a clique, where $A \Delta B$ denotes the symmetric difference of sets A and B. In the cluster editing problem we aim to find a cluster editing set of smallest possible size. In the cluster editing kernelization problem we aim to find a set $X \subset V \times V$, with size as large as possible, such that $X \subset E_{opt}$ for some optimal solution E_{opt} to the cluster editing problem.

2 Solver description

In this paper we provide a short description of the most important algorithms implemented in solver CluES. Due to many parameters used in the implementation and a vast number of case distinctions that need to be taken into account, this description may not contain full information about the algorithms behavior in every possible situation.

Though there are many kernelization rules implemented in CluES (see section 3), the resulting set X of edge modifications returned by the solver is found based on a set of heuristically found cluster editing sets: given cluster editing sets $S_1, \ldots, S_p$ we find the set $X = \bigcap_{i=1}^{p} S_i$ and return it as a final result. It is necessary to mention that the result found this way need not be correct, as the set X may not be contained in any optimum cluster editing set of graph G. Set X will not be a correct result only if all sets S_i contain the same pair $(u, v) \in V \times V$ that does not belong to any optimum solution. By creating a large number of high-quality cluster editing sets and trying to avoid finding similar local optima multiple times, we decrease the probability of returning an incorrect answer. It may also happen that all sets S_i are optimal cluster editing sets, but we will get $X = \emptyset$. In that case, as a resulting set, we can take the largest set of edge modification found by valid kernelization rules described in section 3.

* This is a brief description of one of the highest ranked solvers of PACE Challenge 2021. It has been made public for the benefit of the community and was selected based on the ranking. PACE encourages publication of work building on the ideas presented in this description in peer-reviewed venues.

In the algorithm we take $p = 20$ and for each $i \leq p$ we find the set S_i using the heuristic solver CluES (article "PACE Solver Description: CluES - a heuristic solver for the cluster editing problem" in the current proceedings). We need to add here that sets S_i are found using many different sets of parameters that usually enable us to find cluster editing sets of very high quality. What is more important, by selecting appropriate values for parameters we are often able to find many different local optima to the cluster editing problem. Even if those local optima are not optimal solutions from the global point of view, they are often optimum in some small (local) subgraphs of given graph G. By steering the parameters we try to avoid getting stuck in similar local optima and thus we reduce the risk of returning an incorrect answer set X.

3 Kernelization rules

We use over 15 parameter-independent kernelization rules based on edge cuts and critical cliques. Additionally we use several preprocessing procedures that are not valid kernelization rules (there is no guarantee of preserving the optimality of the solution size after performing those procedures), but are just a reasonable guess that often enables us to quickly and accurately identify nodes that most often belong to the same cluster in some good solution. Those heuristic procedures are also very useful for finding different local optima.

3.1 Valid kernelization rules

Nine of implemented valid kernelization rules are based on concepts related to critical cliques and can be found in [3] and [2].

Next valid kernelization rule is a "similar neighborhoods" rule (see [1]) applied to an unweighted graph. The rule states that, if there exists an edge $(u, v) \in E$ with $N[u] = N[v] \cup \{w\}$ for some $w \notin N[v]$, then there exists an optimal solution with u and v belonging to the same cluster.

Another three valid kernelization rules are specifications of the "almost clique rule" [1] applied for all triangles, K_4's and induced diamonds. The "almost clique rule" states that, if there exists a subset $C \subset V$ such that $G[C]$ is connected and the size of the minimum cut in $G[C]$ is not smaller than $|\{(a, b) \in C \times C : (a, b) \notin E\}| + |\{(a, b) \in E : a \in C, b \notin C\}|$, then there exists an optimal solution where all nodes in C belong to the same cluster.

3.2 Heuristic preprocessing procedures

Now, we proceed to describe heuristic preprocessing procedures. Most of them are based on properties of some intersections of neighborhoods of critical cliques. The following rules depend on miscellaneous parameters and are usually executed multiple times for different values of those parameters.

1. Let $A = \{K_1, \ldots, K_p\}$ be a set of critical cliques in G with the same neighborhood $N(K_i)$ for all $i \leq p$ and let $G' = G[V \setminus \bigcup_{i=1}^{p} K_i]$. If there exists an index $i \leq p$ and a set $K' \subset V$ such that K' is a critical clique in G', $N(K_i) \subset K'$, $|N(K_i)| \leq \alpha \cdot |K'|$ and $|K_i| \leq \beta \cdot |K'|$, where α and β are parameters, then make K_i a single cluster and remove it from G.
2. Let $A = \{u_1, \ldots, u_p\}$ be a set of nodes for which $N(u_i)$ is a clique and let $G' = G[V \setminus A]$. If there exists an index $i \leq p$ and a set $K \subset V$ such that K is a critical clique in G', $N(u_i) \subset K$ and $|N(u_i)| \leq \alpha \cdot |K|$, where α is a parameter, then make u_i a single cluster and remove it from G.

3. Let $A = \{K_1, \ldots, K_p\}$ be a set of critical cliques in G for which $N(K_i)$ is a clique and $\beta_1 \leq |K_i| \leq \beta_2$, where β_1, β_2 are parameters. Take $G' = G[V \setminus \bigcup_{i=1}^{p} K_i]$. If there exists an index $i \leq p$ and a set $K' \subset V$ such that K' is a critical clique in G', $N(K_i) \subset K'$ and $|N(K_i)| \leq \alpha \cdot |K'|$, where α is a parameter, then make K_i a single cluster and remove it from G. This rule is a generalization of the previous rule (the previous rule can be obtained by setting $\beta_1 = \beta_2 = 1$).
4. Let $A = \{u_1, \ldots, u_p\}$ be a set of nodes with the same neighborhood $N(u_i)$. Let $m = \alpha \cdot |A|$, where α is a parameter. Add edges (u_{2i-1}, u_{2i}) for $1 \leq i \leq m$ to graph G. It is important to note here that all added edges are stored and, if at some stage of the main algorithm a cluster editing set S_i is found with a single cluster containing both ends of a stored edge, then the edge is removed from S_i.
5. Let $(u, v) \in E$ be an edge with $N[u] \subset N[v]$, $|N[u]| \geq \alpha \cdot |N[v]|$, $|N(v)| - |N(u)| \leq \beta$ and $|N(u) \cap N(v)| \geq \gamma$. Mark nodes u, v to belong to the same cluster. Marking nodes is done by creating a proper permutation graph before finding cluster editing sets using a heuristic solver CluES. By setting $\alpha = \gamma = 0$, $\beta = 1$ we can obtain the "similar neighborhoods" rule described in 3.1.
6. Let $C = \{v_1, \ldots, v_p\} \subset V$ be a set of nodes for which $G[C]$ is connected. For $v \in C$ let $deg_{out}(v) = |N(v) \setminus C|$. Assume that for parameters α, β and γ the following conditions hold:
   a. $\max_{v \in C} deg_{out}(v) \leq \alpha$
   b. $\sum_{v \in C} deg_{out}(v) \leq \beta$
   c. $\max_{u,v \in C} |(N(u) \cap N(v)) \setminus C| \leq \gamma$

   Then mark all nodes $v \in C$ with $deg_{out}(v) \leq \delta$ (where δ is a parameter) to belong to the same cluster and, depending on values of the parameters, remove from the graph some edges from a set $\{(u, v) \in E : u \in C, v \notin C\}$. This procedure is applied to all triangles, K_4's and induced diamonds. It is also applied to some clusters found by the heuristic solver CluES. Let us mention that knowing the structure of the graph $G[C]$ and selecting proper values of parameters we can obtain the "almost clique rule" described in 3.1.

4 Availability

The source code of CluES is available at `https://doi.org/10.5281/zenodo.4949787` (please select the proper version of the solver to access required contest track: heuristic, exact or kernelization).

References

1 Sebastian Böcker, Sebastian Briesemeister, and Gunnar Klau. Exact algorithms for cluster editing: Evaluation and experiments. *Algorithmica*, 60:316–334, January 2008.

2 Jianer Chen and Jie Meng. A 2k kernel for the cluster editing problem. *Journal of Computer and System Sciences*, 78(1):211–220, 2012. JCSS Knowledge Representation and Reasoning.

3 Jiong Guo. A more effective linear kernelization for cluster editing. *Theor. Comput. Sci.*, 410:718–726, March 2009.