10th International Symposium on Parameterized and Exact Computation

IPEC'15, September 16–18, 2015, Patras, Greece

Edited by

Thore Husfeldt

Iyad Kanj

Editors

Thore Husfeldt
Lund University and ITU Copenhagen
Denmark
thore.husfeldt@cs.lth.se

Iyad Kanj
DePaul University, Chicago
US
ikanj@cs.depaul.edu

ACM Classification 1998
F.1.3 Complexity Measures and Classes, F.2 Analysis of Algorithms and Problem Complexity, G.2 Discrete Mathematics

ISBN 978-3-939897-92-7

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-939897-92-7.

Publication date
November, 2015

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

Digital Object Identifier: 10.4230/LIPIcs.IPEC.2015.i

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

http://www.dagstuhl.de/lipics

Contents

10th International Symposium on Parameterized and Exact Computation (IPEC 2015).
Editors: Thore Husfeldt and Iyad Kanj

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

Preface

This volume contains the papers presented at IPEC 2015: the 10th International Symposium on Parameterized and Exact Computation held during September 16–18, 2015, in Patras, Greece. IPEC was held together with seven other algorithms conferences as part of the annual ALGO congress.

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 IPECs

2004	Bergen, Norway
2006	Zürich, Switzerland
2008	Victoria, Canada
2009	Copenhagen, Denmark
2010	Chennai, India
2011	Saarbrücken, Germany
2012	Ljubljana, Slovenia
2013	Sophia Antipolis, France
2014	Wrocław, Poland

In response to the call for papers, 51 papers were submitted. Each submission was reviewed by at least 3 reviewers. The reviews came from the 14 members of the program committee, and from 56 external reviewers contributing 65 external reviews. The program committee held electronic meetings through the EasyChair.

The program committee felt that the median submission quality was very high, and in the end selected 32 of the submissions for presentation at the symposium and for inclusion in this proceedings volume. This is a record number of talks for IPEC and resulted in three well-attended conference days with a tight schedule of contributed presentations that provided rich opportunities for many researchers to meet and interact. We hope that future symposia can maintain this level both of quality and fruitful interactions.

The program committee presented the IPEC 2015 Excellent Student Paper award to Stefan Kratsch and Manuel Sorge for their paper *On Kernelization and Approximation for the Vector Connectivity Problem*.

IPEC invited two plenary speakers to the ALGO meeting, which served as opportunities to present IPEC-related results and concepts to a wider algorithmic audience. Dimitrios Thilikos spoke on *Bidimensionality and Paramterized Algorithms*, as part of the award ceremony for the 2015 EATCS–IPEC Nerode Prize for outstanding papers in the area of multivariate algorithmics. The award was given by a committee consisting of David Eppstein, Jan Arne Telle, and Georg Gottlob to the authors Eric D. Demaine, Fedor V. Fomin, Mohammadtaghi Hajiaghayi, and Dimitrios M. Thilikos of the paper *Subexponential Parameterized Algorithms on Bounded-Genus Graphs and H-Minor-Free Graphs* [J. ACM **52** (6), 2005]. The other plenary talk was given by Virginia Vassilevska Williams, on *Hardness of Easy Problems: Basing Hardness on Popular Conjectures such as the Strong Exponential Time Hypothesis*, surveying a highly successful line of current research connecting some of the hardness concepts developed in exponential-time complexity to polynomial-time computation. We thank the speakers for accepting our invitation, and for contributing accessible survey papers to this proceedings volume.

The steering committee of IPEC has decided to move the publication of the IPEC proceedings to an open access model. IPEC 2015 is the first year of producing the proceedings under this model. This transition leads to a re-distribution in workload and responsibilities compared to earlier years. While we feel that a large part of the research community supports the move to an open-access model, it has also become clear during the production

10th International Symposium on Parameterized and Exact Computation (IPEC 2015).
Editors: Thore Husfeldt and Iyad Kanj
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of these proceedings that many authors are reluctant to accept the increased demands on invidual responsibility and discipline in paper production that this shift entails. For 2015, the transition significantly increased the workload of the program committee chairs, but we hope that the community can adapt to these changes in the future. We extend our sincere thanks to those authors who invested time to adhere to our instructions for paper production.

We hope that the research presented in the current proceedings will eventually appear in final form in an appropriate journal. In fact, we have invited a number of conributions to IPEC 2015 to a special issue of the journal *Algorithmica*, where we serve as editors.

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. Finally, we are grateful to the local organizers of ALGO, chaired by Christos Zaroliagis, for the efforts, which made chairing IPEC an enjoyable experience.

Thore Husfeldt and Iyad Kanj

Berkeley and Chicago, October 2015

Program Committee

Yijia Chen
Shanghai Jia Tong University

Holger Dell
Saarland University

Fabrizio Grandoni
IDSIA

Jiong Guo
Shandong University

Thore Husfeldt (chair)
ITU Copenhagen and Lund University

Iyad Kanj (chair)
DePaul University

Petteri Kaski
Aalto University

Mikko Koivisto
Helsinki Institute for Information Technology

Stefan Kratsch
TU Berlin

Daniel Lokshtanov
University of Bergen

Rahul Santhanam
The University of Edinburgh

Saket Saurabh
The Institute of Mathematical Sciences

Ioan Todinca
Universite d'Orleans

Magnus Wahlstrom
Royal Holloway University of London

10th International Symposium on Parameterized and Exact Computation (IPEC 2015).
Editors: Thore Husfeldt and Iyad Kanj
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

External Reviewers

Akanksha Agrawal
Yufei Cai
Yixin Cao
Ruiwen Chen
Rajesh Chitnis
Ferdinando Cicalese
Pål Grønås Drange
Markus Sortland Dregi
Andrew Drucker
Michael Elberfeld
Stefan Fafianie
Qilong Feng
Jakub Gajarský
Robert Ganian
Serge Gaspers
Petr Golovach
Danny Hermelin
Bart M. P. Jansen
Mark Jones
Mamadou Moustapha Kanté
Eun Jung Kim
Sudeshna Kolay
Christian Komusiewicz
Janne H. Korhonen
Mithilesh Kumar
Michael Lampis
Paolo Liberatore
Mathieu Liedloff
Bingkai Lin
Fredrik Manne
Dániel Marx
Neeldhara Misra
Matthias Mnich
Pedro Montealegre
Amer Mouawad
Moritz Müller
Sebastian Ordyniak
Fahad Panolan
Christophe Paul
Anthony Perez
Geevarghese Philip
Vuong Anh Quyen
Venkatesh Raman
Peter Rossmanith
Ignasi Sau
Pascal Schweitzer
Christoph Stockhusen
Meesum Syed Mohammad
Xiuting Tao
Dimitrios Thilikos
Erik Jan van Leeuwen
Andreas Wiese
James Worrell
Mingyu Xiao
Meirav Zehavi
Chihao Zhang

List of Authors

Max Bannach
Florian Barbero
Hans L. Bodlaender
Ilario Bonacina
Marthe Bonamy
Édouard Bonnet
Chiel B. ten Brinke
Jannis Bulian
Hubie Chen
Jason Crampton
Anuj Dawar
Holger Dell
Pål Grønås Drange
Eduard Eiben
Andrei Gagarin
Robert Ganian
Archontia C. Giannopoulou
Petr A. Golovach
Gregory Gutin
Pinar Heggernes
Danny Hermelin
Frank J. P. van Houten
Lars Jaffke
Bart M.P. Jansen
Jisu Jeong
Mark Jones
Naonori Kakimura
Mamadou Moustapha Kanté
Moshe Kaspi
Eun Jung Kim
Sudeshna Kolay
Christian Komusiewicz
Łukasz Kowalik
Dieter Kratsch
Stefan Kratsch
Judith-Madeleine Kubitza
O-joung Kwon
Michael Lampis
Daniel Lokshtanov
Tobias Mömke
Diptapriyo Majumdar
Kazuhisa Makino
George B. Mertzios
Valia Mitsou
Barak Navon
Rolf Niedermeier
Fahad Panolan
Christophe Paul
Astrid Pieterse
Geevarghese Philip
Michał Pilipczuk
Venkatesh Raman
Felix Reidl
Clément Requilé
Takayuki Sakai
Fernando Sánchez Villaamil
R. B. Sandeep
Ignasi Sau
Saket Saurabh
Kazuhisa Seto
Dvir Shabtay
Bin Sheng

Somnath Sikdar

Florian Sikora

Naveen Sivadasan

Arkadiusz Socała

Manuel Sorge

Christoph Stockhusen

Ondřej Suchý

Hanna Sumita

Stefan Szeider

Sigve Hortemo Sæther

Navid Talebanfard

Nimrod Talmon

Suguru Tamaki

Till Tantau

Jan Arne Telle

Junichi Teruyama

Dimitrios M. Thilikos

Virginia Vassilevska Williams

Gerhard Woeginger

Tom C. van der Zanden

Bidimensionality and Parameterized Algorithms*

Dimitrios M. Thilikos[1,2,3]

1 **AlGCo Project Team, CNRS, LIRMM, Montpellier, France**
`sedthilk@thilikos.info`
2 **Department of Mathematics, University of Athens, Athens, Greece**
3 **Computer Technology Institute & Press "Diophantus", Patras, Greece**

Abstract

We provide an exposition of the main results of the theory of bidimensionality in parameterized algorithm design. This theory applies to graph problems that are bidimensional in the sense that i) their solution value is not increasing when we take minors or contractions of the input graph and ii) their solution value for the (triangulated) $(k \times k)$-grid graph grows as a quadratic function of k. Under certain additional conditions, mainly of logical and combinatorial nature, such problems admit subexponential parameterized algorithms and linear kernels when their inputs are restricted to certain topologically defined graph classes. We provide all formal definitions and concepts in order to present these results in a rigorous way and in their latest update.

1998 ACM Subject Classification G.2.1 Combinatorics, G.2.2 Graph Theory

Keywords and phrases Parameterized algorithms, Subexponential FPT-algorithms, Kernelization, Linear kenrels, Bidimensionality, Graph Minors

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.1

Category Invited Paper

1 Introduction

The theory of bidimensionality, was introduced in [27] and has been developed further during the last decade in [33, 35, 28, 38, 55, 59, 67, 54, 53] (see also [30, 34, 52, 26, 39, 32]). It provides general techniques for designing efficient fixed-parameter algorithms and approximation schemes for NP-hard graph problems in broad classes of graphs.

A parameterized problem on graphs can be seen as a subset Π of $\mathcal{G} \times \mathbb{N}$ where $\mathcal{G}$ is some graph class (for instance, planar graphs) and the question is whether an instance (G, k) is a member of Π, where k is the parameter of the problem. The main objective is to design an $f(k) \cdot n^{O(1)}$-step algorithm that answers this question while keeping the parametric dependence $f(k)$ as low as possible. This implies that, for each fixed value k, the problem can be solved by an algorithm running in polynomial-time where the degree of this polynomial does not depend on the value of k.

The combinatorial base of bidimensionality is the celebrated grid-exclusion theorem from the Graph Minors series of Robertson and Seymour [82, 81]. This theorem states that every graph excluding a $(r \times r)$-grid as a minor should have treewidth bounded by some function of r (see Subsection 2.1 for the formal definition of treewidth and the minor relation). Treewidth

* The work of this paper was co-financed by the European Union (European Social Fund ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: ARISTEIA II.

10th International Symposium on Parameterized and Exact Computation (IPEC 2015).
Editors: Thore Husfeldt and Iyad Kanj; pp. 1–16

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

is a cornerstone parameter in algorithmic graph theory measuring the topological resemblance of a graph to the structure of a tree.

The central idea of bidimensionality resides in the fact, that for many parameterized graph problems, the presence in their input graphs of a (bidimensional) $(\Omega(\sqrt{k}) \times \Omega(\sqrt{k}))$-grid as a minor is directly providing a positive (or a negative) answer to the problem. The bidimensionality condition, together with certain conditions on $\mathcal{G}$, is able to reduce the problem to the case where the treewidth of the input graph is sublinear in the problem parameter k.

A graph of bounded treewidth can be viewed as a "monodimensional" tree-like structure. According to Courcelle's theorem [21], if the problem is expressible in Monadic Second Order Logic (MSO), then it is possible to process this tree-like structure as the input of a tree automaton that can solve the problem in time that is linear in the size of the input graph. If the parametric dependence of this algorithm can be made singly exponential, the sublinear (on k) treewidth of G yields a parameterized algorithm with subexponential parameterized dependence. This simple reasoning, provides a generic way to design parameterized algorithms with subexponential parametric dependence. In many cases, this provides algorithms running in $2^{O(\sqrt{k})} \cdot n^{O(1)}$ which appears to be the best parametric dependence one may expect, according to the results in [14].

In Section 2 we provide all definitions and theorems that support the above ideas. The concept of bidimensionality is formally defined in Section 3 and in Section 4 we abstract the above methodology into a single theorem on subexponential parameterized algorithms (Theorem 7).

Another, somehow more technical, application of bidimensionality is *kernelization*. A kernelization algorithm for a parameterized graph problem Π is a polynomial-time algorithm that reduces every instance (G, k) to an equivalent one (a kernel) whose size is bounded only by a function of k. When this function is linear on k, we say that Π admits a linear kernel (see Subsection 5 for the formal definitions). Kernelization has been a vibrant field of parameterized complexity and a lot of research has been oriented to the derivation of linear kernels for parameterized problems. Bidimensionality theory has meta-algorithmic applications in the derivation of linear kernels. It follows that, given a parameterized problem $\Pi \subseteq \mathcal{G} \times \mathbb{N}$ where $\mathcal{G}$ satisfies certain (topological) conditions, a linear kernel is automatically derived when Π is bidimensional, is expressible in Counting Monadic Second Order Logic, and satisfies some separability condition. We describe this result in Section 5. For this, we present the basic tools supporting it, namely, the notions of protrusion decomposition and protrusion replacement. We also point out some methodological analogies with the previous case of subexponential algorithms, mainly in what concerns the classification of the required tools into algorithmic and combinatorial ones.

In our exposition we present the contributions of bidimensionality theory to parameterized algorithms in their most general, up to now, version. In contrast to previous surveys on this topic [34, 52], we preferred to insist on the rigorous mathematical formalization of this theory which may require (not only for the unexperienced reader) to go through the definitions of Sections 3 and 4. The exposition concludes by some open problems and further directions in Section 6.

2 Basic concepts

In this section we give some basic definitions that are necessary for the exposition of the rest of the paper.

2.1 Graphs

All graphs in this paper are undirected and without multiple edges or loops. Given a graph G, we use the notation $V(G)$ and $E(G)$ for the vertex set and the edge set of G respectively. We say that a graph H is a *subgraph* of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. Given a set $S \subseteq V(G)$ we denote by $G[S]$ the subgraph G' of G where $V(G') = S$ and $E(G') = \{\{x, y\} \in E(G) \mid \{x, y\} \subseteq S\}$ and we call G' the *subgraph of G induced by S* or we simply say that G' is an induced subgraph of G. Given a set $S \subseteq V(G)$, we denote by $\partial_G(S)$ the set of all vertices in S that are adjacent in G with vertices not in S. We also define the neighborhood of S in G by $N_G(S) = \partial_G(V(G) \setminus S)$.

Treewidth. A *tree decomposition* of a graph G is a pair $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$, where T is a tree whose every node t is assigned a vertex subset $X_t \subseteq V(G)$, called a bag, such that the following three conditions hold:

- $\bigcup_{t \in V(T)} X_t = V(G)$, i.e., every vertex of G is in at least one bag.
- For every $\{u, v\} \in E(G)$, there exists a node t of T such that $u, v \in X_t$.
- For every $u \in V(G)$, the graph $T[\{t \in V(T) \ : \ u \in X_t\}]$ is connected.

The *width* of a tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ equals $\max_{t \in V(T)} |X_t| - 1$. The *treewidth* of a graph G, denoted by $\mathbf{tw}(G)$, is the minimum possible width of a tree decomposition of G.

Minors and contractions. Given an edge $e = \{x, y\}$ of a graph G, the graph G/e is obtained from G by contracting the edge e, that is, the endpoints x and y are replaced by a new vertex $v_{x,y}$ which is adjacent to the old neighbors of x and y (except from x and y). A graph H obtained by a sequence of edge-contractions is said to be a *contraction* of G. We denote it by $H \leq_c G$. A graph H is a *minor* of a graph G if H is the contraction of some subgraph of G and we denote it by $H \leq_m G$. We say that a graph G is *H-minor-free* when it does not contain H as a minor. We also say that a graph class $\mathcal{G}$ is *H-minor-free* (or, excludes H as a minor) when all its members are H-minor-free. A graph G is an *apex graph* if there exists a vertex v such that $G \setminus v$ is planar. A graph class $\mathcal{G}$ is *apex-minor-free* if there exists an apex graph H that is not in $\mathcal{G}$.

Grids and triangulated grids. Given a positive integer k, we denote by $\boxplus_k$ the $(k \times k)$-grid. Formally, for a positive integer k, a *$(k \times k)$-grid* $\boxplus_k$ is a graph with vertex set $\{(x, y) \ : \ x, y \in \{1, \ldots, k\}\}$. Thus $\boxplus_k$ has exactly k^2 vertices. Two different vertices (x, y) and (x', y') are adjacent if and only if $|x - x'| + |y - y'| = 1$.

For an integer $t > 0$, the graph Γ_t is obtained from the grid $\boxplus_t$ by adding, for all $1 \leq x, y \leq t - 1$, the edge $(x+1, y), (x, y+1)$, and additionally making vertex (t, t) adjacent to all the other vertices (x, y) with $x \in \{1, t\}$ or $y \in \{1, t\}$, i.e., to the whole border of $\boxplus_t$. Graph Γ_9 is shown in Fig. 1.

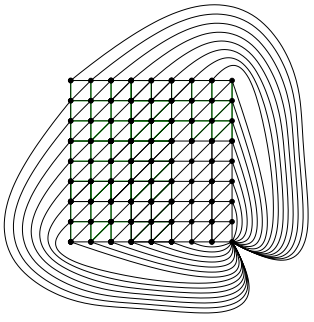


Figure 1 The graph Γ_9.

2.2 Properties of graph classes

A graph class $\mathcal{G}$ is said to be *minor-closed/contraction-closed* if every minor/contraction of a graph in $\mathcal{G}$ also belongs to $\mathcal{G}$.

In general, it is known that there exists a constant c such that any graph G which excludes a $\boxplus_t$ as a minor has treewidth at most $O(t^c)$. The exact value of c remains unknown, but it is more than 2 and at most 36 [15, 20], while it is believed that $c \leq 3$ [36]. We will restrict our attention to graph classes on which $c < 2$ as it is then when bidimensionality theory applies. In particular we say that a graph class $\mathcal{G}$ has the *subquadratic grid minor property* (**SQGM** property for short) if there exist constants $\lambda > 0$ and $1 \leq c < 2$ such that any graph $G \in \mathcal{G}$ which excludes $\boxplus_t$ as a minor has treewidth at most λt^c.

Problems that are contraction-closed but not minor-closed are considered on more restricted classes of graphs. We say that a graph class $\mathcal{G}$ has the *subquadratic gamma contraction* (**SQGC** property for short) if there exist constants $\lambda > 0$ and $1 \leq c < 2$ such that any *connected* graph $G \in \mathcal{G}$ excluding Γ_t as a contraction has treewidth at most λt^c.

The following proposition, for the case of **SQGM**, follows directly from the linearity of excluded grid-minor in H-minor-free graphs proven by Demaine and Hajiaghayi [35], while for the case if **SQGC** it follows from [54].

▶ **Proposition 1.** *For every graph H, H-minor-free graph class $\mathcal{G}$ has the **SQGM** property for some λ depending on H and with $c = 1$. If H is an apex graph, then $\mathcal{G}$ has the **SQGC** property for some λ depending on H and with $c = 1$.*

Notice hat every graph class G with the **SQGC** property has the **SQGM** property. Clearly, the class of planar graphs has both above properties as there is an apex graph containing both K_5 and $K_{3,3}$ as a minor.

Recently, graph classes with the **SQGM** property that are not defined in the context of minor exclusion where detected. In [57] it was proven that unit disk graphs with maximum degree Δ have the **SQGM** property for some λ depending on Δ and with $c = 1/2$. This result has been extended for more general families of geometric intersection graphs in [67].

2.3 Parameterized problems on graphs

Parameterized problems. A parameterized problem Π can be seen as a subset of $\Sigma^* \times \mathbb{N}$ (we denote by $\mathbb{N}$ the set of all non-negative integers). We say that two instances (x, k) and (x', k') of some parameterized problem Π are *equivalent* if and only if $(x, k) \in \Pi \iff (x', k') \in \Pi$.

Parameterized tractable problems. Let Π be a parameterized problem. We say that Π is *fixed parameter tractable* if there exists a function $f : \mathbb{N} \to \mathbb{N}$ and an algorithm deciding whether $(x, k) \in \Pi$ (i.e., whether (x, k) is a YES-instance of Π) in $f(k) \cdot |x|^{O(1)}$ steps. We call such an algorithm FPT-*algorithm*. A parameterized problem belongs to the parameterized class FPT if it can be solved by an FPT-algorithm. (See the monographs [46, 77, 50, 23] on parameterized algorithms and complexity.)

Parameterized graph problems. We say that a parameterized problem Π is a *parameterized graph problem* when in each instance $(x, k) \in \Pi$, x encodes a graph. From now on, we deal with parameterized graph problems as subsets of $\mathcal{G}_{\text{all}} \times \mathbb{N}$ where $\mathcal{G}_{\text{all}}$ is the set of all graphs. Let $\mathcal{G}$ be a class of graphs, i.e, $\mathcal{G} \subseteq \mathcal{G}_{\text{all}}$. The *restriction* of a parameterized problem Π to $\mathcal{G}$ is defined as $\Pi \Cap \mathcal{G} = \{(G, k) \mid (G, k) \in \Pi \text{ and } G \in \mathcal{G}\}$.

2.4 Counting Monadic Second Order Logic

The syntax of Monadic Second Order Logic (MSO) of graphs includes the logical connectives $\vee$, $\wedge$, $\neg$, $\Leftrightarrow$, $\Rightarrow$, variables for vertices, edges, sets of vertices, and sets of edges, the quantifiers $\forall$, $\exists$ that can be applied to these variables, and the following five binary relations:

1. $u \in U$ where u is a vertex variable and U is a vertex set variable;
2. $d \in D$ where d is an edge variable and D is an edge set variable;
3. $\mathbf{inc}(d, u)$, where d is an edge variable, u is a vertex variable, and the interpretation is that the edge d is incident with the vertex u;
4. $\mathbf{adj}(u, v)$, where u and v are vertex variables and the interpretation is that u and v are adjacent;
5. equality of variables representing vertices, edges, sets of vertices, and sets of edges.

In addition to the usual features of monadic second-order logic, if we have atomic formulas testing whether the cardinality of a set is equal to q modulo r, where q and r are integers such that $0 \leq q < r$ and $r \geq 2$, then this extension of the MSO is called *counting monadic second-order logic*. Thus CMSO is MSO enriched with the following atomic formula for a set S: $\mathbf{card}_{q,r}(S) = \mathbf{true}$ if and only if $|S| \equiv q \pmod r$. We refer to [4, 21, 22] for a detailed introduction on CMSO and its algorithmic consequences.

3 Bidimensionality

In this section we define all concepts that are necessary for the definition of the bidimensionality property of parameterized graph problems.

3.1 Subset problems

A *vertex subset* (resp. *edge subset*) *certifying function* ϕ is a computable function which takes as input a graph G and a set $S \subseteq V(G)$ (resp. a set $S \subseteq E(G)$) and outputs true or false.

A *vertex* (resp. *edge*) *subset minimization/maximization problem* Π is a parameterized problem on graphs for which there exists a vertex (resp. edge) certifying function ϕ such that for every $(G, k) \in \mathcal{G} \times \mathbb{N}$ it holds that $(G, k) \in \Pi$ if and only if there exists a set $S \subseteq V(G)$ (resp. $S \subseteq E(G)$) such that $|S| \leq k$ for minimization problems (or $|S| \geq k$ for maximization problems) so that $\phi(G, S) = \mathsf{true}$. If, additionally, there exists a CMSO formula ψ such that $\phi(G, S) = \mathsf{true}$ if and only if $(G, S) \models \psi$, then we say that Π is a MIN-CMSO problem (or a MAX-CMSO problem).

For an example, for the DOMINATING SET problem we have that $\phi(G, S) = \mathsf{true}$ if and only if $\forall v \in V(G)(v \in S \vee \exists u \in V(G) : \mathbf{adj}(v, u))$. Therefore DOMINATING SET is a vertex subset minimization problem that is also as min-CMSO problem.

For simplicity, we will also use the term *subset problems* instead of vertex or edge subset minimization/optimization problems. Let us remark that there are many subset problems which at a first glance do not look as if they could be captured by this definition. An example is the CYCLE PACKING problem. Here the input is a graph G and integer k, and the task is to determine whether G contains k pairwise vertex-disjoint cycles $C_1, C_2, \ldots, C_k$. This is a vertex subset maximization problem because G has k vertex-disjoint cycles if and only if there exists a set $S \subseteq V(G)$ of size at least k and $\phi(G, S)$ is true, where $\phi(G, S)$ is defined such that $\phi(G, S) = \mathsf{true} \iff G$ contains a subgraph G' such that each connected component of G' is a cycle and each connected component of G' contains exactly one vertex from S.

The above definition of CYCLE PACKING may seem bizarre, since checking whether $\phi(G, S)$ is true for a given graph G and set S is NP-complete. In fact this problem is considered as a more difficult problem than CYCLE PACKING. Nevertheless, this definition shows that CYCLE PACKING is indeed a subset problem.

3.2 Optimality functions

For any vertex or edge subset minimization problem Π we have that $(G,k) \in \Pi$ implies that $(G,k') \in \Pi$ for all $k' \geq k$. Similarly, for a vertex or edge subset maximization problem we have that $(G,k) \in \Pi$ implies that $(G,k') \in \Pi$ for all $k' \leq k$. Thus the notion of "optimality" is well defined for subset problems.

▶ **Definition 2.** For a vertex or edge subset minimization problem Π, we define

$$OPT_\Pi(G) = \min\{k \ : \ (G,k) \in \Pi\}.$$

If no k such that $(G,k) \in \Pi$ exists, $OPT_\Pi(G)$ returns $+\infty$. For a vertex or edge subset maximization problem Π,

$$OPT_\Pi(G) = \max\{k \ : \ (G,k) \in \Pi\}.$$

If no k such that $(G,k) \in \Pi$ exists, $OPT_\Pi(G)$ returns $-\infty$. We define $SOL_\Pi(G)$ to be a function that, given as an input a graph G, returns a set S of size $OPT_\Pi(G)$ such that $\phi(G,S) = \mathsf{true}$, and returns **null** if no such set S exists.

▶ **Definition 3.** A subset problem Π is *contraction-closed* (resp. *minor-closed*) if for any two graphs G_1 and G_2 it holds that $G_1 \leq_c G_2 \Rightarrow OPT_\Pi(G_1) \leq OPT_\Pi(G_1)$ (resp. $G_1 \leq_m G_2 \Rightarrow OPT_\Pi(G_1) \leq OPT_\Pi(G_2)$).

3.3 Bidimensional problems

We are now ready to introduce the concept of bidimensionality.

▶ **Definition 4** (Bidimensional problem)**.** A subset problem Π is

minor-bidimensional if
- Π is minor-closed, and
- $\lim_{k\to\infty} \frac{OPT_\Pi(\boxplus_k)}{k^2} = \delta > 0$

contraction-bidimensional if
- Π is contraction-closed, and
- $\lim_{k\to\infty} \frac{OPT_\Pi(\Gamma_k)}{k^2} = \delta > 0$

In each of the above cases (when applicable), we say that the positive real δ is the *density* of the problem Π. A subset problem Π is *bidimensional* if it is minor or contraction bidimensional.

Examples of bidimensional subset problems are (Connected) Vertex Cover, (Connected) Feedback Vertex Set, Induced Matching, Longest Cycle, (Induced) Cycle Packing, d-Scattered Set, Longest Path, (Connected) r-Dominating Set, Diamond Hitting Set, Face Cover, (Connected) Edge Dominating Set, and Unweighted TSP Tour.

It is usually quite easy to determine whether a problem is contraction (or minor) bidimensional. Take as an example Independent Set. Contracting an edge may never increase the size of the maximum independent set, so the problem is contraction-closed. Furthermore it is easy to verify that Γ_k contains an independent set of size $\frac{(k-1)^2}{4}$. Thus Independent Set is contraction-bidimensional with density $1/4$. On the other hand deleting edges may increase the size of a maximum-size independent set in G. Thus Independent Set is not minor-bidimensional.

4 Subexponential parameterized algorithms

A central problem in parameterized algorithm design is to investigate in which cases and under which input restrictions a parameterized problem belongs to FPT and, if so, to find algorithms with the simplest possible parameter dependence.

Let Π be a parameterized graph problem in FPT that can be solved in $f(k) \cdot n^{O(1)}$ steps[1]. When $f(k) = 2^{o(k)}$ we say that Π admits a *subexponential parameterized algorithm* (see [44] for a survey on subexponential parameterized algorithms).

In [14], Cai and Juedes proved that several parameterized problems do not admit subexponential parameterized algorithms, unless 3-SAT can be solved in time subexponential in the number of its variables[2]. Among them, one can distinguish core problems such as the standard parameterizations of Vertex Cover, Dominating Set, and Feedback Vertex Set. However, it appears that many parameterized problems admit subexponential parameterized algorithms running in $2^{O(\sqrt{k})} \cdot n^{O(1)}$ steps when their inputs are restricted to planar graphs or other classes of surface embeddable graphs. Moreover, the results of [14] indicated that the parameterized dependence $2^{O(\sqrt{k})}$ is the best we may expect when the planarity restriction is imposed. The first subexponential parameterized algorithm on planar graphs appeared in [1] for Dominating Set, Independent Dominating Set, and Face Cover. After that, subexponential parameterized algorithms where designed for many other problems [85, 31, 1, 71, 19, 29, 48, 49, 70, 62, 72, 37, 24, 58]. Most of these results are now covered by the main result of this section (Theorem 7).

4.1 Singly exponentially solvable problems w.r.t. treewidth

Let Π be a subset problem Π. We say that Π is *singly exponentially solvable with respect to treewidth* if there exists an algorithm that computes $OPT_\Pi(G)$ in $2^{O(\mathbf{tw}(G))}n^{O(1)}$ steps.

Typically, to prove that a subset problem Π is singly exponentially solvable with respect to treewidth requires the design of dynamic programming algorithms on tree decompositions of width at most w whose tables are of singly exponential size on w. The design of such algorithms has occupied a lot of research in parameterized complexity [8, 87, 83, 13, 3, 5, 40, 6, 25, 45, 42, 43, 84]. In most of the cases, such algorithms run in $2^{O(\mathbf{tw}(G))}n$ steps. A general meta-algorithmic condition implying that a problem is singly exponentially solvable with respect to treewidth was given in [78] and is a model of Modal Logic called *Existential Counting Modal Logic* (ECM-Logic).

4.2 Bidimensionality and subexponential parameterized algorithms

Let Π be a vertex/edge subset minimization (resp. maximization) problem. Consider the following two conditions for Π.

A [*Algorithmic*] Π is singly exponentially solvable with respect to treewidth.

B [*Combinatorial*] If (G, k) is a YES- (resp. NO-) instance of Π, then $\mathbf{tw}(G) = o(k)$.

▶ **Proposition 5.** *If Π is a vertex/edge subset minimization (resp. maximization) problem satisfying conditions **A** and **B**, then Π admits a subexponential parameterized algorithm.*

Proof. Let (G, k) be an input for Π. If the treewidth of the input graph exceeds the upper bound of the combinatorial condition **B**, then we can safely report that (G, k) is a NO- (resp.

[1] From now on, we use n to denote the number of vertices of the input graph G, i.e., $n = |V(G)|$.

[2] This is hypothesis is also known as the Exponential Time Hypothesis (ETH).

YES-) instance and we are done. This step can be supported by the algorithm in [9] that, given a graph G and an integer w, either returns that $\mathbf{tw}(G) > w$, or outputs a tree-decomposition of G of width $\leq 5w$. If now the bound of the combinatorial condition **B** holds, we have a tree-decomposition of G of width $5 \cdot \mathbf{tw}(G) = o(k)$ and the result follows directly from the algorithmic condition **A**. ◀

The following is an important combinatorial consequence of bidimensionality. It reflects the original idea of [27].

▶ **Proposition 6.** *If Π is a subset problem that is minor (resp. contraction) bidimensional and $\mathcal{G}$ is a graph class with the **SQGM** (resp. **SQGC**) property, then $\Pi \Cap \mathcal{G}$ satisfies the combinatorial condition **B**.*

Proof. We give the proof in the case where Π is a vertex/edge subset minimization problem. For this, we have to show that if (G, k) is a YES-instance of $\Pi \Cap \mathcal{G}$, then $\mathbf{tw}(G) = o(k)$. Indeed, if $(G, k) \in \Pi$ then

$$OPT_\Pi(G) \leq k. \tag{1}$$

$$\text{If } \boxplus_r \leq_m G, \text{ then } \quad OPT_\Pi(\boxplus_r) \leq OPT_\Pi(G). \tag{2}$$

As Π is minor (resp. contraction) bidimensional, then

$$OPT_\Pi(\boxplus_r) = \Omega(r^2). \tag{3}$$

From (1), (2), and (3), it follows that if $\boxplus_r \leq_m G$, then $r = O(\sqrt{k})$ which, from the **SQGM** (resp. **SQGC**) property of $\mathcal{G}$ implies that $\mathbf{tw}(G) = o(k)$. ◀

Using Propositions 5 and 6, we easily conclude with the following.

▶ **Theorem 7.** *Let Π be a vertex/edge subset minimization (resp. maximization) problem that*

- **i.** *is singly exponentially solvable with respect to treewidth and*
- **ii.** *is minor- (resp. contraction-) bidimensional*

*and let $\mathcal{G}$ be a graph class with the **SQGM** (resp. **SQGC**) property. Then the restriction of Π to $\mathcal{G}$ admits a* subexponential parameterized algorithm*.*

Notice that the above theorem can become purely meta-algorithmic if we replace condition *i.* by the expressibility of Π in ECM-Logic, as indicated by the results in [78]. Clearly, for the applicability of the above approach, it is important to detect graph classes with the **SQGM** (resp. **SQGC**) property. Historically, this was first done for bounded genus graphs in [27] and in [38], for H-minor free graphs in [35], [28], and [54], and for families of geometric graphs in [57] and [67]. Finally, results that either use ideas similar to bidimensionality o provide alternative techniques for the derivation of subexponential (or low-exponential) parameterized algorithms have been examined in [47, 56, 79, 41, 80, 66].

5 Kernelization

Kernelization has been extensively studied in parameterized complexity. It can be seen as the strategy of analyzing preprocessing or data reduction routines from a parameterized complexity perspective.

5.1 Kernelization algorithms

The notion of *kernelization* is formally defined as follows.

▶ **Definition 8.** A *kernelization algorithm*, or simply a *kernel*, for a parameterized problem Π is an algorithm $\mathcal{A}$ that, given an instance (x,k) of Π, runs in polynomial, on $|x|$, time and outputs an equivalent instance (x',k') of Π where $|x'|+k' \leq g(k)$ for some computable function $g : \mathbb{N} \to \mathbb{N}$ called the *size* of the kernel. In this case we say that Π admits a g *kernel* and if the size g is a polynomial (resp. linear) function of the parameter k, then we say that Π admits a *polynomial (resp. linear) kernel*. As we agreed for parameterized graph problems, we will assume that x corresponds to a graph and we treat the size of a kernel as a function on the number of vertices of the graph in the equivalent instance.

Notable examples of known kernels are a $2k$ kernel for Vertex Cover [18], a $355k$ kernel for Dominating Set on planar graphs [2], which later was improved to a $67k$ kernel [17] and an $O(k^2)$ kernel for Feedback Vertex Set [86] parameterized by the solution size. One of the most intensively studied directions in kernelization is the study of problems on planar graphs and other classes of sparse graphs. This study was initiated by Alber et al. [2] who gave the first linear-sized kernel for the Dominating Set problem on planar graphs. The work of Alber et al. [2] triggered an explosion of papers on kernelization, and kernels of linear sizes were obtained for a variety of parameterized problems on planar graphs including Connected Vertex Cover, Minimum Edge Dominating Set, Maximum Triangle Packing, Efficient Edge Dominating Set, Induced Matching, Full-Degree Spanning Tree, Feedback Vertex Set, Cycle Packing, Blue-Red Dominating Set, and Connected Dominating Set [2, 11, 12, 17, 51, 68, 69, 75, 76, 64, 60]. Most of these results are now covered by the main result of this section (Theorem 13). We refer to the surveys [73, 74] for a detailed exposition of the area of kernelization.

5.2 Separability

We now restrict our attention to problems Π that are somewhat well-behaved in the sense that whenever we have a small separator in the graph that splits the graph in two parts L and R, the intersection $|X \cap L|$ of L with any optimal solution X to the entire graph is a good estimate of $OPT_\Pi(G[L])$. This behavior is called *separability* and variants of it have been used, combined with bidimensionality, for the derivation of Efficient Polynomial Time Approximation Schemes (EPTAS), see [33, 55].

▶ **Definition 9** (Separability)**.** Let $f : \mathbb{N} \to \mathbb{N}$. We say that a subset problem Π is *f-separable* if for any graph G and $L \subseteq V(G)$ such that $|\partial_G(L)| \leq t$, it holds that

$$|SOL_\Pi(G) \cap L| - f(t) \leq OPT_\Pi(G[L]) \leq |SOL_\Pi(G) \cap L| + f(t).$$

Π is called *separable* if there exists a function f such that Π is *f-separable*. Π is called *linearly separable* if it is *f-separable* for some linear function f.

5.3 Protrusion decompositions and replacements

We introduce the notions of *protrusion*, *protrusion decomposition*, and *protrusion replacement*.

Protrusion decompositions. Given a graph G, we say that a set $X \subseteq V(G)$ is an t-protrusion of G if $|\partial(X)| \leq t$ and $\mathbf{tw}(G[X]) \leq t$. An *(α,β)-protrusion decomposition* of a graph G is a partition $\mathcal{P} = \{R_0, R_1, \ldots, R_\rho\}$ of $V(G)$ such that

- $\max\{\rho, |R_0|\} \leq \alpha$,
- each $R_i^+ = N_G[R_i]$, $i \in \{1, \ldots, \rho\}$, is a β-protrusion of G, and
- for every $i \in \{1, \ldots, \rho\}$, $N_G(R_i) \subseteq R_0$.

Protrusion replacement algorithms. Let Π be a parameterized graph problem and let $f : \mathbb{Z}^+ \to \mathbb{Z}^+$ be a non-decreasing function. An *f-protrusion replacement family* for Π is a collection $\mathcal{A} = \{\mathsf{A}_i \mid i \geq 0\}$ of algorithms, such that algorithm A_i receives as input a pair (I, X), where $I = (G, k)$ is an instance of Π and X is an i-protrusion of G with at least $f(i)$ vertices and outputs an equivalent instance $I^* = (G^*, k^*)$ where $|V(G^*)| < |V(G)|$ and $k^* \leq k$. We say that Π has a *protrusion replacement family* if it has a f-protrusion replacement family for some $f : \mathbb{Z}^+ \to \mathbb{Z}^+$.

5.4 Meta-algorithmic results for kernels

Let Π be a vertex/edge subset minimization (resp. maximization) problem. The following two conditions for such a problem Π were defined in [10, 7]. They can be seen as the "kernelization counterparts" of the properties **A** and **B** that we introduced in Subsection 4.2.

A [*Algorithmic*] Π has a protrusion replacement family.

B [*Combinatorial*] If (G, k) is a YES- (resp. NO-) instance of Π, then G has an $(O(k), O(1))$-protrusion decomposition.

The next result is a special case of Theorem 4.6 in [10, 7].

▶ **Proposition 10.** *If a parameterized graph problem Π has properties **A** and **B**, then Π admits a linear kernel.*

The following result is based on the property that a problem has Finite Integer Index (FII). In [10, Lemma 5.19] it was proved that this problem property is able to yield property **A** and, as it has recently been proved in [61], FII is a consequence of CMSO expressibility and the separability property.

▶ **Proposition 11.** *Every* MIN/MAX-CMSO *subset problem Π that is linearly separable has property **A**.*

We now present one of the main combinatorial consequences of bidimensionality. It has been proved in [59, 61].

▶ **Proposition 12.** *Let $\mathcal{G}$ be a graph class with the **SQGM** (resp. **SQGC**) property and let Π be a subset problem that is minor- (resp. contraction-) bidimensional and linear-separable. Then $\Pi \Cap \mathcal{G}$ satisfies property **B**.*

Using Propositions 10, 11, and 12, one can easily derive the following meta-algorithmic result.

▶ **Theorem 13.** *Let Π be a subset problem that*

i. *is a* MIN/MAX-CMSO *problem,*
ii. *is minor- (resp. contraction-) bidimensional,*
iii. *is linearly separable,*

*and let $\mathcal{G}$ is a graph class with the **SQGM** (resp. **SQGC**) property. Then the restriction of Π to $\mathcal{G}$ admits a linear kernel.*

6 Further extensions

In this paper we presented two consequences of bidimensionality, namely Theorem 7 (subexponential parameterized algorithms) and 13 (linear kernelization). Further applications of bidimensionality on the automatic derivation of EPTAS can be found in [33] and [55]. It is an interesting question whether this problem property can be exploited to other algorithmic paradigms.

For the existing applications, there are two main directions. The first is to enlarge the set of graph classes satisfying the **SQGM** or the **SQGC** property. A first step, escaping from the graph minors framework, are the results of [57] and [67] on geometric graphs. Another direction is to make the constants involved in Theorems 7 and 13 explicit so as to optimize the running times of the derived algorithms. A step is this direction was taken in [63] using dynamic programming for certain families of problems. It is also interesting to build extensions of bidimensionality for problems that instead of being closed under minors or contractions are closed under some other partial ordering on graphs such as topological minors, immersions, induced minors and others. We believe that recent results such as those in [65, 88, 16] might be helpful starting points in this direction.

Acknowledgments. I am thankful to Fedor V. Fomin, Stavros G. Kolliopoulos, and Spyros Maniatis, for their helpful remarks on the manuscript.

References

1 J. Alber, H. L. Bodlaender, H. Fernau, T. Kloks, and R. Niedermeier. Fixed parameter algorithms for dominating set and related problems on planar graphs. *Algorithmica*, 33(4):461–493, 2002.

2 J. Alber, M. R. Fellows, and R. Niedermeier. Polynomial-time data reduction for dominating sets. *J. ACM*, 51:363–384, 2004.

3 J. Alber and R. Niedermeier. Improved tree decomposition based algorithms for domination-like problems. In S. Rajsbaum, editor, *LATIN 2002: Theoretical Informatics*, volume 2286 of *LNCS*, pages 221–233. Springer Berlin / Heidelberg, 2002.

4 S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12:308–340, 1991.

5 N. Betzler, R. Niedermeier, and J. Uhlmann. Tree decompositions of graphs: Saving memory in dynamic programming. *Discrete Optimization*, 3(3):220 – 229, 2006. Graphs and Combinatorial Optimization.

6 A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Fourier meets Möbius: fast subset convolution. In *STOC*, pages 67–74. ACM, 2007.

7 H. Bodlaender, F. Fomin, D. Lokshtanov, E. Penninkx, S. Saurabh, and D. Thilikos. (Meta) kernelization. In *50th Annual IEEE Symposium on Foundations of Computer Science, (FOCS 2009)*. ACM, 2009.

8 H. L. Bodlaender. Dynamic programming on graphs with bounded treewidth. In *15th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 317 of *LNCS*, pages 105–118. Springer-Verlag, 1988.

9 H. L. Bodlaender, P. G. Drange, M. S. Dregi, F. V. Fomin, D. Lokshtanov, and M. Pilipczuk. An $O(c^k n)$ 5-approximation algorithm for treewidth. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 499–508, 2013.

10 H. L. Bodlaender, F. V. Fomin, D. Lokshtanov, E. Penninkx, S. Saurabh, and D. M. Thilikos. (meta) kernelization. *CoRR*, abs/0904.0727, 2009.

11 H. L. Bodlaender and E. Penninkx. A linear kernel for planar feedback vertex set. In *Proccedings of the 3rd International Workshop on Exact and Parameterized Computation (IWPEC 2008)*, volume 5018 of *LNCS*, pages 160–171. Springer, Berlin, 2008.

12 H. L. Bodlaender, E. Penninkx, and R. B. Tan. A linear kernel for the k-disjoint cycle problem on planar graphs. In *19th International Symposium on Algorithms and Computation (ISAAC)*, volume 5369 of *LNCS*, pages 306–317. Springer, 2008.

13 H. L. Bodlaender and J. A. Telle. Space-efficient construction variants of dynamic programming. *Nordic J. Comput.*, 11(4):374–385, 2004.

14 L. Cai and D. Juedes. On the existence of subexponential parameterized algorithms. *J. Comput. System Sci.*, 67(4):789 – 807, 2003.

15 C. Chekuri and J. Chuzhoy. Polynomial bounds for the grid-minor theorem. *CoRR*, abs/1305.6577, 2013.

16 C. Chekuri and J. Chuzhoy. Degree-3 treewidth sparsifiers. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 242–255, 2015.

17 J. Chen, H. Fernau, I. A. Kanj, and G. Xia. Parametric duality and kernelization: Lower bounds and upper bounds on kernel size. *SIAM Journal on Computing*, 37:1077–1106, 2007.

18 J. Chen, I. A. Kanj, and W. Jia. Vertex cover: further observations and further improvements. *J. Algorithms*, 41(2):280–301, 2001.

19 J. Chen, I. A. Kanj, L. Perković, E. Sedgwick, and G. Xia. Genus characterizes the complexity of certain graph problems: Some tight results. *Journal of Computer and System Sciences*, 73(6):892 – 907, 2007.

20 J. Chuzhoy. Excluded grid theorem: Improved and simplified. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, STOC '15, pages 645–654, New York, NY, USA, 2015. ACM.

21 B. Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990.

22 B. Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. *Handbook of Graph Grammars*, pages 313–400, 1997.

23 M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.

24 M. Cygan, S. Kratsch, and J. Nederlof. Fast hamiltonicity checking via bases of perfect matchings. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 301–310, New York, NY, USA, 2013. ACM.

25 M. Cygan, J. Nederlof, M. Pilipczuk, M. Pilipczuk, J. M. M. van Rooij, and J. O. Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, (FOCS 2011)*, pages 150–159. IEEE Computer Society, 2011.

26 E. D. Demaine. Algorithmic graph minors and bidimensionality. In *Graph Theoretic Concepts in Computer Science - 36th International Workshop, WG 2010, Zarós, Crete, Greece, June 28-30, 2010 Revised Papers*, page 2, 2010.

27 E. D. Demaine, F. V. Fomin, M. Hajiaghayi, and D. M. Thilikos. Subexponential parameterized algorithms on graphs of bounded-genus and H-minor-free graphs. In *Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 830–839 (electronic). ACM, New York, 2004.

28 E. D. Demaine, F. V. Fomin, M. Hajiaghayi, and D. M. Thilikos. Bidimensional parameters and local treewidth. *SIAM J. Discrete Math.*, 18(3):501–511, 2005.

29 E. D. Demaine, F. V. Fomin, M. Hajiaghayi, and D. M. Thilikos. Fixed-parameter algorithms for (k, r)-center in planar graphs and map graphs. *ACM Trans. Algorithms*, 1(1):33–47, 2005.

30 E. D. Demaine, F. V. Fomin, M. Hajiaghayi, and D. M. Thilikos. Bidimensional structures: Algorithms, combinatorics and logic (dagstuhl seminar 13121). *Dagstuhl Reports*, 3(3):51–74, 2013.

31 E. D. Demaine, F. V. Fomin, M. T. Hajiaghayi, and D. M. Thilikos. Fixed-parameter algorithms for (k, r)-center in planar graphs and map graphs. *ACM Transactions on Algorithms*, 1(1):33–47, 2005.

32 E. D. Demaine and M. Hajiaghayi. Bidimensionality, map graphs, and grid minors. *CoRR*, abs/cs/0502070, 2005.

33 E. D. Demaine and M. Hajiaghayi. Bidimensionality: new connections between fpt algorithms and ptass. In *16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2005)*, pages 590–601. ACM-SIAM, New York, 2005.

34 E. D. Demaine and M. Hajiaghayi. The bidimensionality theory and its algorithmic applications. *Comput. J.*, 51(3):292–302, 2008.

35 E. D. Demaine and M. Hajiaghayi. Linearity of grid minors in treewidth with applications through bidimensionality. *Combinatorica*, 28(1):19–36, 2008.

36 E. D. Demaine, M. Hajiaghayi, and K. Kawarabayashi. Algorithmic graph minor theory: Improved grid minor bounds and Wagner's contraction. *Algorithmica*, 54(2):142–180, 2009.

37 E. D. Demaine, M. Hajiaghayi, and D. M. Thilikos. Exponential speedup of fixed-parameter algorithms for classes of graphs excluding single-crossing graphs as minors. *Algorithmica*, 41:245–267, 2005.

38 E. D. Demaine, M. Hajiaghayi, and D. M. Thilikos. The Bidimensional Theory of Bounded-Genus Graphs. *SIAM Journal on Discrete Mathematics*, 20(2):357–371, 2006.

39 E. D. Demaine and M. T. Hajiaghayi. Fast algorithms for hard graph problems: Bidimensionality, minors, and local treewidth. In *Graph Drawing, 12th International Symposium, GD 2004, New York, NY, USA, September 29 - October 2, 2004, Revised Selected Papers*, pages 517–533, 2004.

40 F. Dorn. Dynamic programming and fast matrix multiplication. In *14th Annual European Symposium on Algorithms (ESA 2006)*, volume 4168 of *LNCS*, pages 280–291. Springer, Berlin, 2006.

41 F. Dorn, F. V. Fomin, D. Lokshtanov, V. Raman, and S. Saurabh. Beyond bidimensionality: Parameterized subexponential algorithms on directed graphs. *Information and Computation*, 233(0):60 – 70, 2013.

42 F. Dorn, F. V. Fomin, and D. M. Thilikos. Fast subexponential algorithm for non-local problems on graphs of bounded genus. In *10th Scandinavian Workshop on Algorithm Theory (SWAT 2006)*, volume 4059 of *LNCS*, pages 172–183. Springer, Berlin, 2006.

43 F. Dorn, F. V. Fomin, and D. M. Thilikos. Catalan structures and dynamic programming in H-minor-free graphs. In *ACM-SIAM Symposium on Discrete Algorithms (SODA 2008)*, pages 631–640. SIAM, 2008.

44 F. Dorn, F. V. Fomin, and D. M. Thilikos. Subexponential parameterized algorithms. *Computer Science Review*, 2(1):29–39, 2008.

45 F. Dorn, E. Penninkx, H. L. Bodlaender, and F. V. Fomin. Efficient exact algorithms on planar graphs: Exploiting sphere cut decompositions. *Algorithmica*, 58(3):790–810, 2010.

46 R. G. Downey and M. R. Fellows. *Parameterized complexity.* Monographs in Computer Science. Springer-Verlag, New York, 1999.

47 F. F. Dragan, F. V. Fomin, and P. A. Golovach. Spanners in sparse graphs. In *35th International Colloquium on Automata, Languages and Programming (ICALP 2008)*, volume 5125 of *LNCS*, pages 597–608. Springer, 2008.

48 H. Fernau. Graph separator algorithms: a refined analysis. In *Graph-theoretic Concepts in Computer Science*, volume 2573 of *LNCS*, pages 186–197. Springer, Berlin, 2002.

49 H. Fernau and D. Juedes. A geometric approach to parameterized algorithms for domination problems on planar graphs. In *29th International Symposium on Mathematical Foundations of Computer (MFCS 2004)*, volume 3153 of *LNCS*, pages 488–499. Springer, Berlin, 2004.

50 J. Flum and M. Grohe. *Parameterized Complexity theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2006.

51 F. V. Fomin, S. S. Daniel Lokshtanov, and D. M. Thilikos. Linear kernels for (connected) dominating set on H-minor-free graphs. In *23st ACM–SIAM Symposium on Discrete Algorithms (SODA 2012)*. ACM-SIAM, San Francisco, California, 2012.

52 F. V. Fomin, E. D. Demaine, and M. T. Hajiaghayi. Bidimensionality. In *Encyclopedia of Algorithms*. Springer, 2015.

53 F. V. Fomin, P. Golovach, and D. M. Thilikos. Contraction bidimensionality: the accurate picture. In *17th Annual European Symposium on Algorithms (ESA 2009)*, LNCS, pages 706–717. Springer, 2009.

54 F. V. Fomin, P. A. Golovach, and D. M. Thilikos. Contraction obstructions for treewidth. *J. Comb. Theory, Ser. B*, 101(5):302–314, 2011.

55 F. V. Fomin, D. Lokshtanov, V. Raman, and S. Saurabh. Bidimensionality and EPTAS. *CoRR*, abs/1005.5449, 2010.

56 F. V. Fomin, D. Lokshtanov, V. Raman, and S. Saurabh. Subexponential algorithms for partial cover problems. *Inf. Process. Lett.*, 111(16):814–818, 2011.

57 F. V. Fomin, D. Lokshtanov, and S. Saurabh. Bidimensionality and geometric graphs. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1563–1575, 2012.

58 F. V. Fomin, D. Lokshtanov, and S. Saurabh. Efficient computation of representative sets with applications in parameterized and exact algorithms. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pages 142–151. SIAM, 2014.

59 F. V. Fomin, D. Lokshtanov, S. Saurabh, and D. M. Thilikos. Bidimensionality and kernels. In *21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2010)*, pages 503–510. ACM-SIAM, 2010.

60 F. V. Fomin, D. Lokshtanov, S. Saurabh, and D. M. Thilikos. Linear kernels for (connected) dominating set on graphs with excluded topological subgraphs. In N. Portier and T. Wilke, editors, *30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013)*, volume 20 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 92–103, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

61 F. V. Fomin, D. Lokshtanov, S. Saurabh, and D. M. Thilikos. Bidimensionality and kernels. Revised manuscript, 2015.

62 F. V. Fomin and D. M. Thilikos. Dominating sets in planar graphs: branch-width and exponential speed-up. *SIAM J. Comput.*, 36(2):281–309 (electronic), 2006.

63 V. Garnero, C. Paul, I. Sau, and D. M. Thilikos. Explicit linear kernels via dynamic programming. In *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014), STACS 2014, March 5-8, 2014, Lyon, France*, pages 312–324, 2014.

64 V. Garnero, I. Sau, and D. M. Thilikos. A linear kernel for planar red-blue dominating set. In *Proceedings of the 12th Cologne Twente Workshop on Graphs and Combinatorial Optimization (CTW)*, pages 117–120, Enschede, The Netherlands, May 2013.

65 A. C. Giannopoulou and D. M. Thilikos. Optimizing the graph minors weak structure theorem. *SIAM J. Discrete Math.*, 27(3):1209–1227, 2013.

66 P. A. Golovach, M. Kamiński, D. Paulusma, and D. M. Thilikos. Induced packing of odd cycles in a planar graph. In *20th International Symposium on Algorithms and Computation (ISAAC 2009)*, volume 5878 of *LNCS*, pages 514–523. Springer, Berlin, 2009.

67 A. Grigoriev, A. Koutsonas, and D. Thilikos. Bidimensionality of geometric intersection graphs. In V. Geffert, B. Preneel, B. Rovan, J. Štuller, and A. Tjoa, editors, *SOFSEM 2014: Theory and Practice of Computer Science*, volume 8327 of *Lecture Notes in Computer Science*, pages 293–305. Springer International Publishing, 2014.

68 J. Guo, R. Niedermeier, and S. Wernicke. Fixed-parameter tractability results for full-degree spanning tree and its dual. In *Second International Workshop on Parameterized and Exact Computation (IWPEC)*, volume 4169 of *LNCS*, pages 203–214. Springer, 2006.

69 I. Kanj, M. J. Pelsmajer, M. Schaefer, and G. Xia. On the induced matching problem. *Journal of Computer and System Sciences*, 77(6):1058 – 1070, 2011.

70 I. Kanj and L. Perković. Improved parameterized algorithms for planar dominating set. In *27th International Symposium Mathematical Foundations of Computer Science (MFCS 2002)*, volume 2420 of *LNCS*, pages 399–410. Springer, Berlin, 2002.

71 T. Kloks, C. M. Lee, and J. Liu. New algorithms for k-face cover, k-feedback vertex set, and k-disjoint cycles on plane and planar graphs. In *28th International Workshop on Graph Theoretic Concepts in Computer Science (WG 2002)*, volume 2573 of *LNCS*, pages 282–295. Springer, Berlin, 2002.

72 A. Koutsonas and D. M. Thilikos. Planar feedback vertex set and face cover: Combinatorial bounds and subexponential algorithms. *Algorithmica*, 60(4):987–1003, 2011.

73 S. Kratsch. Recent developments in kernelization: A survey. *Bulletin of the EATCS*, 113, 2014.

74 D. Lokshtanov, N. Misra, and S. 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.

75 D. Lokshtanov, M. Mnich, and S. Saurabh. Linear kernel for planar connected dominating set. In *6th Annual Conference on Theory and Applications of Models of Computation (TAMC 2009)*, volume 5532 of *LNCS*, pages 281–290. Springer, 2009.

76 H. Moser and S. Sikdar. The parameterized complexity of the induced matching problem in planar graphs. In *Proceedings First Annual International WorkshopFrontiers in Algorithmics (FAW)*, volume 4613 of *LNCS*, pages 325–336. Springer, 2007.

77 R. Niedermeier. *Invitation to fixed-parameter algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, Oxford, 2006.

78 M. Pilipczuk. Problems parameterized by treewidth tractable in single exponential time: A logical approach. In F. Murlak and P. Sankowski, editors, *Mathematical Foundations of Computer Science 2011*, volume 6907 of *Lecture Notes in Computer Science*, pages 520–531. Springer Berlin Heidelberg, 2011.

79 M. Pilipczuk, M. Pilipczuk, P. Sankowski, and E. J. van Leeuwen. Subexponential-time parameterized algorithm for steiner tree on planar graphs. In *30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013, February 27 - March 2, 2013, Kiel, Germany*, pages 353–364, 2013.

80 M. Pilipczuk, M. Pilipczuk, P. Sankowski, and E. J. van Leeuwen. Network sparsification for steiner problems on planar and bounded-genus graphs. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 276–285, 2014.

81 N. Robertson and P. D. Seymour. Graph Minors. V. Excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 41(2):92–114, 1986.

82 N. Robertson, P. D. Seymour, and R. Thomas. Quickly excluding a planar graph. *J. Combin. Theory Ser. B*, 62(2):323–348, 1994.

83 J. Rué, I. Sau, and D. M. Thilikos. Dynamic programming for graphs on surfaces. In *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010 (1)*, volume 6198 of *Lecture Notes in Computer Science*, pages 372–383. Springer, 2010.

84 J. Rué, I. Sau, and D. M. Thilikos. Asymptotic enumeration of non-crossing partitions on surfaces. *Discrete Mathematics*, 313(5):635–649, 2013.

85 D. M. Thilikos. Fast sub-exponential algorithms and compactness in planar graphs. In *19th Annual European Symposium on Algorithms (ESA 2011)*, pages 358–369, 2011.

86 S. Thomassé. A $4 \cdot k^2$ kernel for feedback vertex set. *ACM Trans. Algorithms*, 6(2):32:1–32:8, Apr. 2010.

87 J. M. M. van Rooij, H. L. Bodlaender, and P. Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In *ESA*, pages 566–577. Springer-Verlag, 2009.

88 P. Wollan. The structure of graphs not admitting a fixed immersion. *J. Comb. Theory, Ser. B*, 110:47–66, 2015.

Hardness of Easy Problems: Basing Hardness on Popular Conjectures such as the Strong Exponential Time Hypothesis

Virginia V. Williams*

Stanford University, Computer Science Department, Stanford, CA, USA

Abstract

Algorithmic research strives to develop fast algorithms for fundamental problems. Despite its many successes, however, many problems still do not have very efficient algorithms. For years researchers have explained the hardness for key problems by proving NP-hardness, utilizing polynomial time reductions to base the hardness of key problems on the famous conjecture P $\neq$ NP. For problems that already have polynomial time algorithms, however, it does not seem that one can show any sort of hardness based on P $\neq$ NP. Nevertheless, we would like to provide evidence that a problem A with a running time $O(n^k)$ that has not been improved in decades, also *requires* $n^{k-o(1)}$ time, thus explaining the lack of progress on the problem. Such unconditional time lower bounds seem very difficult to obtain, unfortunately. Recent work has concentrated on an approach mimicking NP-hardness: (1) select a few key problems that are conjectured to require $T(n)$ time to solve, (2) use special, fine-grained reductions to prove time lower bounds for many diverse problems in P based on the conjectured hardness of the key problems. In this abstract we outline the approach, give some examples of hardness results based on the Strong Exponential Time Hypothesis, and present an overview of some of the recent work on the topic.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, F.1.3 Complexity Measures and Classes

Keywords and phrases reductions, satisfiability, strong exponential time hypothesis, shortest paths, 3SUM, equivalences, fine-grained complexity

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.17

Category Invited Paper

1 Introduction

The core goal of algorithmic research is to determine how fast computational problems can be solved in the worst case. Important time hierarchy theorems from complexity theory imply that (for natural models of computation) for all time-constructible functions $T(n)$, there are problems that can be solved in $T(n)$ time but not in $O(T(n)^{1-\varepsilon})$ time for $\varepsilon > 0$. The main challenge is to determine where in this hierarchy various natural and fundamental problems lie. Throughout the years, many ingenious algorithmic techniques have been developed and applied to obtain blazingly fast algorithms for many important problems. For instance, many useful graph problems can be solved in near-linear time (*e.g.*, finding connected components or computing single source shortest paths). Recent breakthroughs have shown that problems like graph matching, linear programming and max flow have surprisingly fast algorithms as

* This work was partially supported by NSF Grants CCF-1417238 and CCF-1514339, and BSF Grant BSF:2012338.

10th International Symposium on Parameterized and Exact Computation (IPEC 2015).
Editors: Thore Husfeldt and Iyad Kanj; pp. 17–29

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

well ([46, 42, 43]). Nevertheless, for many other central problems the best known running times are essentially those of the classical algorithms devised for them in the 1950s and 1960s.

A prominent example is the CNF-SAT problem: given a boolean formula on n variables in conjunctive normal form, determine whether it has a satisfying assignment. The naïve algorithm for the problem is to try all possible 2^n assignments to the variables and check whether they satisfy the clause. This algorithm runs in $O^*(2^n)$ time, where O^* hides polynomial factors in the number of variables and clauses. When the maximum clause length is a constant k, CNF-SAT is called k-SAT. This problem can be solved in $O^*(2^{n-cn/k})$ for various constants c independent of n and k (*e.g.*, [36, 48, 51, 50, 56, 57]), thus improving over the 2^n running time exponentially. Nevertheless, as k grows, this running time approaches 2^n. Thus the naïve algorithm is essentially the best known algorithm for CNF-SAT even when the clause length is (an arbitrary) constant.

The theory of NP-hardness has been the main tool for explaining why problems such as CNF-SAT are hard. The seminal result that k-SAT is NP-complete for all $k \geq 3$ [41] implies that if we believe that $P \neq NP$, k-SAT cannot have a polynomial time algorithm. NP-hardness however, does not say anything about running times that are not polynomial, and it seems difficult, if not impossible, to show that the 2^n time algorithm for CNF-SAT is optimal, assuming only $P \neq NP$. The optimality of the 2^n time algorithm for CNF-SAT is currently only a conjecture, known as the Strong Exponential Time Hypothesis (SETH).

The type of hardness that SETH asserts about CNF-SAT is not unique to NP-hard problems. Although for the purposes of complexity theory, problems in the class P are considered "easy", this is not because anyone believes that running times such as $O(n^{100})$ or even $O(n^3)$ are efficient. P was initially studied mainly because polynomials have useful properties, *e.g.*, they are closed under composition and also polynomial running times allow us model independence. It became interesting to distinguish the problems in P from those that require superpolynomial time, and questions such as P vs NP emerged.

There are many important problems within P that have (often brute-force) classical algorithms running in $\tilde{O}(n^k)$ time for some constant k,[1] but whose running time has not been improved upon except (possibly) by $n^{o(1)}$ factors. Some prominent examples of such problems from different areas of computer science include: (1) from graph algorithms: the center of a graph G, *i.e.*, $\arg\min_{v \in G} \max_{x \in G} \mathrm{dist}(v, x)$, can be computed in $O(n^3)$ time using Floyd–Warshall's classical algorithm for All-Pairs Shortest Paths, and no faster algorithm is known; (2) from computational biology: given two length n DNA sequences, their longest common subsequence (LCS) can be computed in $O(n^2)$ time using a classical dynamic programming algorithm, and the fastest known algorithm runs faster but only by logarithmic factors [47, 16, 35]; (3) from computational geometry, given n points in the plane, one can determine whether they are in general position with a simple classical algorithm in $\tilde{O}(n^2)$ time, and this is the best known, up to $n^{o(1)}$ factors.

Unconditional lower bounds seem very difficult to obtain – it is not even known whether SAT can be solved in linear time. Mimicking NP-hardness, we would like to give evidence that for problems like the above, the classical algorithms are probably optimal, and that the polynomial time solvable problem is "hard". NP-hardness itself seems to have little use in showing that problems that already have polynomial time algorithms are hard. Instead, a more *fine-grained* approach has been used in recent years. In this approach, we pick a widely believed hypothesis about the time complexity of a *key problem*, and then use *fine-grained* reductions to reduce this key problem to other important problems, giving conditional lower bounds on how fast these problems can be solved.

[1] Here, $\tilde{O}$ hides $n^{o(1)}$ factors.

2 The key problems

The majority of conditional hardness results for problems within P are based on the conjectured hardness of three problems: 3SUM, All-Pairs Shortest Paths, and CNF-SAT. Since we are focusing on exact running times, we need to fix the model of computation. Here we assume that we are working with a Word RAM model with $O(\log n)$ bit words.

2.1 3SUM

The *3SUM* problem asks to determine whether a given set of n integers contains three integers a, b, c so that $a+b=c$. The problem has a simple $\tilde{O}(n^2)$ time algorithm: sort the integers, and for every pair a, b, check whether their sum is in the list. Baran, Demaine and Pătraşcu [14] showed that in the Word RAM model with $O(\log n)$ bit words, 3SUM can be solved in $O(n^3(\log\log n)^2/\log^2 n)$ time, thus obtaining a speed-up[2]. Chan and Lewenstein [20] showed that some interesting structured instances of 3SUM *can* be solved in $O(n^{1.9})$ time. However, there are no known $O(n^{2-\varepsilon})$ time (so-called "truly subquadratic") algorithms for the general problem for any $\varepsilon > 0$, even when randomization can be used. The lack of progress on the problem has led to the following conjecture [53, 32].

▶ **Conjecture 1** (No truly subquadratic 3SUM)**.** In the Word RAM model with $O(\log n)$ bit words, any algorithm requires $n^{2-o(1)}$ time in expectation to determine whether a set $S \subset \{-n^3, \ldots, n^3\}$ of size n contains three distinct elements $a, b, c \in S$ with $a + b = c$.

(By standard hashing arguments, one can assume that the size of the integers in the 3SUM instance is $O(n^3)$, and so the conjecture is not for a restricted version of the problem.)

The 3SUM conjecture is widely believed, especially in computational geometry. In 1995, Gajentaan and Overmars [32] formed a theory of "3SUM-hard problems" by showing that one can reduce 3SUM to many problems in computational geometry and that the conjecture above implies that none of these problems have truly subquadratic algorithms. One example of a 3SUM-hard problem is the planar points in general position problem that we mentioned earlier. Following [32] many other papers proved the 3SUM hardness of geometric problems, *e.g.*, [28, 45, 29, 10, 30, 12, 22, 15]. Vassilevska Williams and Williams [58, 59] showed that a certain weighted graph triangle problem cannot be found efficiently unless Conjecture 1 is false, relating 3SUM to problems in weighted graphs. Their work was extended [4] for other weighted subgraph problems. The 3SUM conjecture has also recently been shown to imply hardness for various problems on strings such as jumbled indexing [11] and versions of sequence local alignment [7]. Pătraşcu [53] initiated the research of proving lower bounds on dynamic problems based on Conjecture 1. He showed that for several dynamic problems Conjecture 1 implies update time lower bounds that are *polynomial* in the input size, whereas the best (unconditional) cell probe lower bounds known are polylogarithmic at best. Follow-up work by [5] extended and tightened Pătraşcu's results.

2.2 All-Pairs Shortest Paths

The *All-Pairs Shortest Paths* problem (APSP) is as follows: given a directed or undirected graph with integer edge weights, determine the distances between every pair of vertices in the graph. Classical algorithms such as Floyd–Warshall's provide $O(n^3)$ running times for

[2] When the inputs are real numbers, the problem can also be sped up by a logarithmic factor [40].

APSP in n-node graphs. For years, researchers obtained larger and larger *polylogarithmic* improvements over the cubic running time, until in a breakthrough result Williams [64] designed an algorithm that runs faster than $O(n^3/(\log n)^c)$ time for all constants c; the exact running time is $n^3/\exp(\Theta(\sqrt{\log n}))$. Nevertheless, no truly subcubic time ($O(n^{3-\varepsilon})$ for $\varepsilon > 0$) algorithm for APSP is known. This led to the following conjecture assumed in many papers, *e.g.* [55, 61].

▶ **Conjecture 2** (No truly subcubic APSP)**.** There is a constant c, such that in the Word RAM model with $O(\log n)$ bit words, any algorithm requires $n^{3-o(1)}$ time in expectation to compute the distances between every pair of vertices in an n node graph with edge weights in $\{1, \ldots, n^c\}$.

Vassilevska Williams and Williams [61] and Abboud, Grandoni and Vassilevska Williams [3] showed that many other graph problems are equivalent to APSP under subcubic reductions, and as a consequence any truly subcubic algorithm for them would violate Conjecture 2. Some examples of these problems include detecting a negative weight triangle in a graph, computing replacement paths and finding the radius of a graph. The APSP conjecture implies strong lower bounds for dynamic problems [55, 5], *e.g.*, for single source shortest paths even if the algorithm is required to support only insertions and deletions.

2.3 Satisfiability

Here we formally describe the Strong Exponential Time Hypothesis (SETH) that we discussed in the introduction. Impagliazzo, Paturi, and Zane [37, 38] introduced SETH to address the question of how fast one can solve k-SAT as k grows. They define:

$$s_k = \inf\{\, \delta \mid \text{there is a } O^*(2^{\delta n}) \text{ time algorithm for } k\text{-SAT with } n \text{ variables} \,\}.$$

The sequence s_k is clearly nondecreasing. As the best known algorithms for k-SAT have running times that converge to $O^*(2^n)$ as k grows, it is natural to conjecture that $\lim_{k\to\infty} s_k = 1$, which is exactly what SETH hypothesizes.

▶ **Conjecture 3** (SETH)**.** For every $\varepsilon > 0$, there exists an integer k, such that Satisfiability on k-CNF formulas on n variables cannot be solved in $O(2^{(1-\varepsilon)n} \operatorname{poly} n)$ time in expectation.

SETH is an extremely popular conjecture in the exact exponential-time algorithms community. For instance, Cygan et al. [24] showed that the SETH is also equivalent to the assumption that several other NP-hard problems cannot be solved faster than by exhaustive search, and the best algorithms for these problems are the exhaustive search ones. Some other work that proves conditional lower bounds based on SETH for NP-hard problems includes [24, 18, 27, 44, 26, 65, 52, 23, 25, 31]. SETH has been the basis for many conditional hardness results for problems in P, including edit-distance [13], LCS [2, 17], graph diameter [54, 21] and many others.

2.4 Orthogonal Vectors

Many hardness results based on SETH go through an intermediate problem, *Orthogonal Vectors* (OV). In this problem, we are given two sets U and V of n vectors each over $\{0,1\}^d$ where $d = \omega(\log n)$, and want to determine whether there are $u \in U$ and $v \in V$ such that $\sum_{i=1}^{d} u_i \cdot v_i = 0$. An equivalent version of the problem has $U = V$.

The naïve algorithm for the problem runs in $O(n^2 d) \leq \tilde{O}(n^2)$ time, and the best known algorithm [9] runs slightly faster, in $O(n^{2-1/O(\log(d/\log n))})$ time. Obtaining a truly subquadratic algorithm for OV has been elusive, and Williams [63] showed that SETH implies the nonexistence of such an algorithm. (We will see the proof of this later.) No reduction is known from OV to CNF-SAT, and the OV problem may require essentially quadratic time even if SETH is false. Thus it is often better to base hardness on the following OV conjecture.

▶ **Conjecture 4** (OVC). In the Word RAM model with $O(\log n)$ bit words, any algorithm requires $n^{2-o(1)}$ time in expectation to determine whether a set of n vectors over $\{0,1\}^d$ for $d = \omega(\log n)$ contains an orthogonal pair.

2.5 The Small Universe Hitting Set Problem

A version of the Orthogonal Vectors problem is the *Small Universe Hitting Set* (HS) problem: given two sets U and V of n sets each over the universe $\{1,\ldots,d\}$ where $d = \omega(\log n)$, determine whether there is a $u \in U$ that hits every $v \in V$ in at least one element. The HS problem can be described analogously as, given two sets U and V of vectors over $\{0,1\}^d$ for $d = \omega(\log n)$, is there a $u \in U$ such that for all $v \in V$, we have $\sum_{i=1}^{d} u_i \cdot v_i > 0$.

Just as with OV, the fastest known algorithm for HS runs in $n^{2-o(1)}$ time. Similar to OVC, there is a conjecture concerning the HS problem:

▶ **Conjecture 5** (HSC). In the Word RAM model with $O(\log n)$ bit words, any algorithm, given two sets U and V of n vectors each over $\{0,1\}^d$ for $d = \omega(\log n)$, requires $n^{2-o(1)}$ time in expectation to determine if U contains a vector that is not orthogonal to any vector in V.

We will show that HSC implies OVC and is thus potentially stronger. An example hardness result based on HSC is that computing the radius of a sparse graph requires $n^{2-o(1)}$ time. Such a result does not seem to be possible based on OVC.

2.6 Relationships between the conjectures

Besides the reduction from CNF-SAT to OV by Williams [63], there are no known reductions between the main key problems. Potentially, any subset of the first three key conjectures could be true while the others are false. Recent work [19] gives evidence that it might be difficult to reduce these conjectures to one another, showing that if for instance one could reduce OV to 3SUM or APSP in a tight way, then a nondeterministic version of SETH would fail. It seems to be difficult to solve SAT quickly even by an algorithm that can exploit nondeterminism since showing that the formula is unsatisfiable seems to require a lot of nondeterministic time. Nevertheless, in recent work, Williams [62] has shown that if the nondeterministic algorithm can use randomness, k-SAT can be solved (for all k), in $O((2-\varepsilon)^n)$ time for a constant $\varepsilon > 0$. Derandomizing Williams' algorithm seems challenging, however, so that the nondeterministic version of SETH might still hold.

Even though there are no known (tight) reductions between CNF-SAT, 3SUM and APSP, there are two known problems that one can reduce all three problems to.

The first problem, *Matching Triangles*, takes as an input an integer Δ, a graph $G = (V, E)$ on n nodes, and a coloring of the nodes $c : V \to \{1,\ldots,n\}$. The problem asks, is there a triple of colors, $a, b, c \in \{1,\ldots,n\}$, such that the number of triangles in G that have node colors a, b, c, is at least Δ.

The second problem, *Triangle Collection*, takes as an input a graph $G = (V, E)$ on n nodes, and a coloring of the nodes $c : V \to \{1,\ldots,n\}$, and asks whether there there is a

triple of colors, $a, b, c \in \{1, \ldots, n\}$, such that there are no triangles in G that have node colors a, b, c. (An alternative name for this problem might be Triangle-Free Color Triple.)

Both Triangle Collection and Matching Triangles can be solved in $O(n^3)$ time by enumerating all triangles in the input graph. A surprising result [8] is that if either of these problems has an $O(n^{3-\varepsilon})$ time algorithm for $\varepsilon > 0$, then the 3SUM, APSP and SETH conjectures are all false. The result holds for Matching Triangles even when Δ is polylogarithmic, and it also holds for a restricted version of Triangle Collection called *Triangle Collection*$*$. The latter problem has been used as a basis for hardness for several problems in dynamic algorithms, such as maintaining the strongly connected components of a graph, and for a few static problems related to Max Flow.

3 The reductions

To prove tight reductions between the running times for solving problems in P, one cannot merely use polynomial time reductions since these do not distinguish between different polynomial running times. Instead, we define *fine-grained reductions* that for problems A and B and running times $a(n), b(n)$ imply that an $O(b(n)^{1-\varepsilon})$ time algorithm for B for constant $\varepsilon > 0$ would imply an $O(a(n)^{1-\delta})$ algorithm for A for constant δ. Notice that we allow δ to be different from ε. This is fine since we merely want to know when any improvement for B carries over to some improvement for A. Having δ and ε differ gives us much more freedom in designing reductions.

An initial attempt would be to say that given an instance of A, the reduction should transform it into a single instance of B, *i.e.*, giving a many-one reduction. This, however, limits us quite a bit. For instance, we wouldn't be able to show that a multi-output problem such as APSP can be reduced to a decision problem such as whether the radius of the graph is less than R. Instead, we define the reductions as special Turing reductions, *i.e.*, each instance of A can be reduced to multiple instances of B.

▶ **Definition 6** (Fine-grained reduction)**.** Let $a(n)$ and $b(n)$ be nondecreasing functions of n. Problem A is (a, b)-reducible to problem B, denoted as $A \leq_{a,b} B$, if for every $\varepsilon > 0$, there is a $\delta > 0$, an algorithm F with access to an oracle to B, a constant d, and for every $n \geq 1$ an integer $k(n)$, such that for every n, the algorithm F takes any instance of A of size n and

- F runs in at most $d \cdot (a(n))^{1-\delta}$ time,
- F produces at most $k(n)$ instances of B adaptively, that is, the jth instance B_j is a function of $\{(B_i, a_i)\}_{1 \leq i < j}$ where B_i is the ith instance produced and a_i is the answer of the oracle for B on instance B_i, and
- the sizes n_i of the instances B_i for any choice of oracle answers a_i obey the inequality $\sum_{i=1}^{k(n)} (b(n_i))^{1-\varepsilon} \leq d \cdot (a(n))^{1-\delta}$.

An immediate consequence of the reductions is that improvements over $b(n)$ for B imply improvements over $a(n)$ for A. More formally, if there is an algorithm for B of running time $c \cdot N^{1-\varepsilon}$ on instances of size N, then composing the algorithm with a fine-grained (a, b)-reduction from A to B produces an algorithm for A running in time

$$d \cdot (a(n))^{1-\delta} + \sum_{i=1}^{k(n)} c \cdot (b(n_i))^{1-\varepsilon} \leq d \cdot (c+1) \cdot (a(n))^{1-\delta},$$

where the first summand is for running the reduction algorithm F, and the second summand is for running the algorithm for B on the instances $\{B_i\}$. As c and d are constants, the running time for A is $O(a(n)^{1-\delta})$.

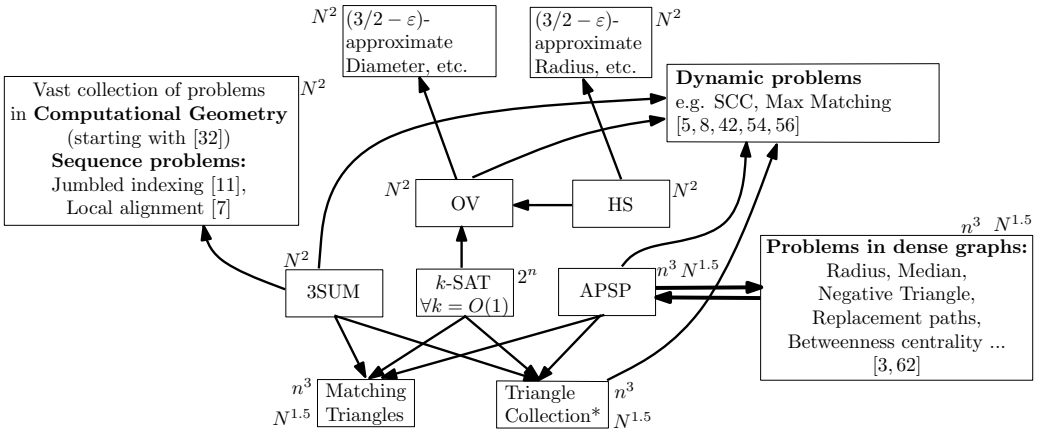


Figure 1 Partial summary of the implications of the main conjectures. An arrow from problem A to problem B, where A has $a(n)$ next to it, B has $b(n)$ next to it, implies that $A \leq_{a,b} B$. When the inputs are graphs, n stands for the number of nodes. N always stands for the total input size. When both n and N are present for a problem, we assume that $N = n^2$; the meaning is that the reductions are only for dense graphs in which case the input size is quadratic in n. For k-SAT, n denotes the number of variables. For the dynamic problems, different key problems can be reduced to different key problems, and the update/query time tradeoffs vary. References are not comprehensive.

Notice that by the above definition, the instance sizes may depend on n and ε, and d can depend on ε, but not on n.

4 Example results

A partial summary of the implications of the main conjectures (3SUM, OV, SETH and APSP) can be found in Figure 1. Many of the reductions in the known results are quite intricate. Here we will present a few simple proofs to illustrate the approach.

We begin with the starting point of many of the reductions from SETH to problems in P. This is Williams' reduction from CNF-SAT to OV.

▶ **Theorem 7** (k-SAT $\leq_{2^n,n^2}$ OV [63]). *For each positive integer k, the k-SAT problem on n variables is $(2^n, n^2)$-reducible to Orthogonal Vectors on n vectors.*

Proof. Let F be a k-SAT formula on n variables. Using the sparsification lemma [38], we can assume that F has $m = O(n)$ clauses, as this only gives a $2^{\alpha n}$ overhead for arbitrarily small $\alpha > 0$. Create two sets U_1, U_2 of vectors over $\{0,1\}^m$. Split the variables into two sets V_1 and V_2 of size $n/2$ each. For each $i \in \{1,2\}$ and each partial assignment ϕ to the variables of V_i, add a vector $v_{i,\phi}$ to U_i where $v_{i,\phi}[c] = 1$ if ϕ does not satisfy clause c. Now, $v_{0,\phi}$ and $v_{1,\psi}$ are orthogonal if and only if for every clause c at least one of ϕ and ψ satisfies clause c. Thus, there is an orthogonal pair if and only if ϕ and ψ together form a satisfying assignment. The number of vectors in the instances created is $N = O(2^{n/2})$ and the number of coordinates is $O(n) = O(\log N)$. ◀

Due to the use of the sparsification lemma, the above reduction produces an exponential number of instances. However, the instances do not depend on the answers of the OV oracle. Below we give a different reduction that is actually adaptive and each new instance depends on the answers to the previous ones.

▶ **Theorem 8** (HS $\leq_{n^2,n^2}$ OV [9, 6]). *Small-Universe Hitting Set on n vectors is (n^2, n^2)-reducible to Orthogonal Vectors on n vectors.*

Proof. Let U, V be an instance of HS where $|U| = |V| = n$. Let s be a parameter to be set later. Partition U into s sets $U_1, \ldots, U_s$ of size at most $\lceil n/s \rceil \leq 2n/s$ each. Similarly, partition V into $V_1, \ldots, V_s$ of size $\leq 2n/s$.

Now, for every choice of $i, j \in \{1, \ldots, s\}$ in turn: while (U_i, V_j) contains an orthogonal pair (u, v), remove u from U (and hence from all U_k) and ask about (U_i, V_j) again; if no orthogonal pair is found, continue to the next choice of (i, j).

If at the end of this procedure U contains some u, then u must be nonorthogonal to all vectors in V, and hence the HS instance is a "yes" instance. Otherwise, if U is empty, then every $u \in U$ was orthogonal to some $v \in V$ and the HS instance is a "no" instance.

The running time is as follows: every call to the OV problem either returns an orthogonal pair (u, v) or determines that no such pair exists in $U_i \times V_j$. The number of times an orthogonal pair can be returned is at most n since when (u, v) is discovered, u is removed from U. On the other hand, each (U_i, V_j) instance can be a "no"-instance of OV at most once, so that the number of calls that return a "no" is at most s^2. Thus, if we set $s = \sqrt{n}$, the number of instances created of OV is at most $2n$ and their sizes are all at most $2\sqrt{n}$. Hence, for every $\varepsilon > 0$, there is a fine-grained reduction that creates at most $2n$ instances whose sizes n_i obey $\sum_{i=1}^{2n} n_i^{2-\varepsilon} \leq 4 \sum_{i=1}^{2n} n^{1-\varepsilon/2} = 8n^{2-\varepsilon/2}$. The reduction only does simple partitioning of the instance and removals from U, and hence runs in $O(n^{2-\varepsilon/2})$ time. ◀

Finally, we give a reduction from OV to the problem of approximating the diameter of a sparse graph, *i.e.*, the largest distance. There is a $\frac{3}{2}$-approximation algorithm for diameter that runs in $\tilde{O}(n^{3/2})$ time in n-node, $\tilde{O}(n)$-edge graphs [54, 21]. In such graphs, the diameter can computed exactly in $\tilde{O}(n^2)$ time by computing APSP. The theorem below shows that, assuming OVC (or SETH), if you want a $(\frac{3}{2} - \varepsilon)$-approximation, you might as well compute all pairwise distances in the graph.

▶ **Theorem 9** (OV $\leq_{n^2,n^2}$ $(3/2 - \varepsilon)$−Diameter [54]). *Orthogonal Vectors on n vectors is (n^2, n^2)-reducible to distinguishing between diameter 2 and 3 in a graph with n nodes and $O(n)$ edges.*

Proof. Let U, V be an instance of OV. For each $u \in U$ create a node u (abusing notation) and for each $v \in V$ create a node v. For each coordinate c, create a node c. For any vector node x (in U or V), add an edge from x to any coordinate node c if and only if $x[c] = 1$. Add two extra nodes a and b with an edge between them. Add an edge from all $u \in U$ to a and from b to all $v \in V$. Now, all pairs of nodes except those in $U \times V$ are at distance at most 2. Two nodes $u \in U$ and $v \in V$ are at distance 2 if there is some c for which $u[c] = v[c] = 1$, and are at distance 3 otherwise, via (a, b). Thus the diameter is 3 if there is an orthogonal pair and is 2 otherwise. ◀

5 Some open problems

5.1 Connections to exact exponential-time algorithms

Many problems in P have fine-grained reductions from CNF-SAT. There are other NP-hard problems, however, that are not known to be equivalent (in the fine-grained sense) to CNF-SAT.

What problems in P do other NP-hard problems reduce to?

Recent work by Abboud et al. [1] gives fine-grained reductions from k-Clique (for any constant k) to two problems in cubic time, CFG recognition and RNA folding. When k is divisible by 3, the fastest known algorithm for k-Clique runs in $n^{\omega k/3+o(k)}$ time [49] where $\omega < 2.373$ is the matrix multiplication exponent [60, 33]. Via the reduction from [1], the conjecture that this algorithm is optimal implies $n^{\omega-o(1)}$ conditional lower bounds for both RNA folding and CFG recognition. Valiant showed in the 1970s that the latter problem can be solved in $O(n^\omega)$ time, and hence this running time is tight assuming the k-clique conjecture.

Williams [63] gave a fine-grained (c^n, n^k)-reduction from any 2-CSP, *i.e.*, any constraint satisfaction problem with at most two variables per constraint (such as MAX-CUT or MAX-2-SAT) to k-Clique for any constant $k \geq 3$. Here c^n represents the brute-force running time for the 2-CSP problem, and for MAX-CUT and MAX-2-SAT, $c = 2$. Since one can find a triangle (*i.e.*, a 3-clique) in a graph in $O(n^{2.373})$ time [39], this immediately gave faster than brute-force algorithms for these problems. Due to Williams' reduction, one can view the result of Abboud et al. [1] as reducing any 2-CSP to CFG recognition and RNA folding. The fine-grained reduction implies that any improvement over the current algorithms for the latter two problems would improve upon the best known algorithms for solving 2-CSPs. What other problems can one reduce 2-CSPs to? What can we reduce Set Cover, TSP, etc. to?

5.2 Connections to fixed parameter tractability

The fixed parameter tractability community studies which NP-hard problems, when parameterized by some parameter k of the input, can be solved in $f(k) \operatorname{poly} n$ time, where n is the size of the input. The idea is that while we believe that NP-hard problems cannot be solved in polynomial time, the hardness can be pushed into some parameter k, so that for all constant values of k, the running time is the same polynomial, independent of k. In a sense, $f(k)$ can be thought of as the constant factor overhead in the running time.

This idea does not have to be restricted to NP-hard problems. Consider a problem that can be solved in $O(n^c)$ time for some constant c and is believed to also require $n^{c-o(1)}$ time. Then, we can ask, for what parameters k is this problem in $f(k)n^{c-\varepsilon}$ time for $\varepsilon > 0$? This question was asked independently by Giannopoulou et al. [34] and Abboud et al. [6]. Giannopoulou et al. consider the Longest Path on interval graphs that is known to be solvable in $O(n^4)$ time. They show that when parameterized by the vertex deletion number k to proper interval graphs, the problem has an $O(k^9 n)$ time algorithm. Abboud et al. explored the diameter problem in sparse graphs which is solvable in $O(n^2)$ time. They developed a fixed parameter subquadratic $2^{O(t \log t)} n$ time algorithm, where t is the treewidth of the input graph, also showing an almost matching lower bound, under SETH.

Under what parameters are APSP, 3SUM, OV in fixed-parameter linear time?

The above question is of course interesting for all hard problems in P.

References

1 Amir Abboud, Arturs Backurs, and Virginia V. Williams. If the current clique algorithms are optimal, so is Valiant's parser. In *56th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, October 17-20, 2015*, page to appear, 2015.

2 Amir Abboud, Arturs Backurs, and Virginia V. Williams. Quadratic-time hardness of LCS and other sequence similarity measures. In *56th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, October 17-20, 2015*, page to appear, 2015.

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, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1681–1697, 2015.

4 Amir Abboud and Kevin Lewi. Exact weight subgraphs and the k-sum conjecture. In *Automata, Languages, and Programming – 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, pages 1–12, 2013.

5 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, Philadelphia, PA, USA, October 18-21, 2014*, pages 434–443, 2014.

6 Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, page to appear, 2016.

7 Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *Automata, Languages, and Programming – 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 39–51, 2014.

8 Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 41–50, 2015.

9 Amir Abboud, Richard 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, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 218–230, 2015.

10 Manuel Abellanas, Ferran Hurtado, Christian Icking, Rolf Klein, Elmar Langetepe, Lihong Ma, Belén Palop, and Vera Sacristán. Smallest color-spanning objects. In *Algorithms – ESA 2001, 9th Annual European Symposium, Aarhus, Denmark, August 28-31, 2001, Proceedings*, pages 278–289, 2001.

11 Amihood Amir, Timothy M. Chan, Moshe Lewenstein, and Noa Lewenstein. On hardness of jumbled indexing. In *Automata, Languages, and Programming – 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 114–125, 2014.

12 Boris Aronov and Sariel Har-Peled. On approximating the depth and related problems. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005*, pages 886–894, 2005.

13 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 on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 51–58, 2015.

14 I. Baran, E.D. Demaine, and M. Pătraşcu. Subquadratic algorithms for 3SUM. *Algorithmica*, 50(4):584–596, 2008.

15 Gill Barequet and Sariel Har-Peled. Polygon-containment and translational min-Hausdorff-distance between segment sets are 3SUM-hard. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, 17-19 January 1999, Baltimore, Maryland.*, pages 862–863, 1999.

16 Philip Bille and Martin Farach-Colton. Fast and compact regular expression matching. *Theoretical Computer Science*, 409(3):486–496, 2008.

17 Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *56th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, October 17-20, 2015*, page to appear, 2015.

18 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In *Parameterized and Exact Computation*, pages 75–85. Springer, 2009.

19 Marco L. Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mikhailin, Ramamohan Paturi, and Stefan Schneider. Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility. Technical Report TR15-148, Electronic Colloquium on Computational Complexity (ECCC), 2015.

20 Timothy M. Chan and Moshe Lewenstein. Clustered integer 3sum via additive combinatorics. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 31–40, 2015.

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

22 Otfried Cheong, Alon Efrat, and Sariel Har-Peled. On finding a guard that sees most and a shop that sells most. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*, pages 1098–1107, 2004.

23 Marek Cygan. Deterministic parameterized connected vertex cover. In *Algorithm Theory – SWAT 2012 – 13th Scandinavian Symposium and Workshops, Helsinki, Finland, July 4-6, 2012. Proceedings*, pages 95–106, 2012.

24 Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as CNF-SAT. In *Proceedings of the 27th Conference on Computational Complexity, CCC 2012, Porto, Portugal, June 26-29, 2012*, pages 74–84, 2012.

25 Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast Hamiltonicity checking via bases of perfect matchings. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 301–310, 2013.

26 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 *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 150–159, 2011.

27 Evgeny Dantsin and Alexander Wolpert. On moderately exponential time for SAT. In *Proc. 13th International Conference on Theory and Applications of Satisfiability Testing*, pages 313–325, 2010.

28 Mark de Berg, Marko de Groot, and Mark H. Overmars. Perfect binary space partitions. *Computational Geometry: Theory and Applications*, 7(81):81–91, 1997.

29 J. Erickson. New lower bounds for convex hull problems in odd dimensions. *SIAM Journal on Computing*, 28(4):1198–1214, 1999.

30 William S. Evans, Daniel Archambault, and David G. Kirkpatrick. Computing the set of all distant horizons of a terrain. In *Proceedings of the 16th Canadian Conference on Computational Geometry, CCCG'04, Concordia University, Montréal, Québec, Canada, August 9-11, 2004*, pages 76–79, 2004.

31 Henning Fernau, Pinar Heggernes, and Yngve Villanger. A multivariate analysis of some DFA problems. In *Language and Automata Theory and Applications – 7th International Conference, LATA 2013, Bilbao, Spain, April 2-5, 2013. Proceedings*, pages 275–286, 2013.

32 A. Gajentaan and M. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Computational Geometry*, 5(3):165–185, 1995.

33 François Le Gall. Powers of tensors and fast matrix multiplication. In *International Symposium on Symbolic and Algebraic Computation, ISSAC'14, Kobe, Japan, July 23-25, 2014*, pages 296–303, 2014.

34 Archontia C. Giannopoulou, George B. Mertzios, and Rolf Niedermeier. Polynomial fixed-parameter algorithms: A case study for longest path on interval graphs. *CoRR*, abs/1506.01652, 2015.

35 Szymon Grabowski. New tabulation and sparse dynamic programming based techniques for sequence similarity problems. In *Stringology*, pages 202–211, 2014.

36 Edward A. Hirsch. Two new upper bounds for SAT. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, 25-27 January 1998, San Francisco, California.*, pages 521–530, 1998.

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

38 R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

39 A. Itai and M. Rodeh. Finding a minimum circuit in a graph. *SIAM J. Computing*, 7(4):413–423, 1978.

40 Allan Grønlund Jørgensen and Seth Pettie. Threesomes, degenerates, and love triangles. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 621–630, 2014.

41 Richard 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.*, pages 85–103, 1972.

42 Yin Tat Lee and Aaron Sidford. Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 147–156, 2013.

43 Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in $\tilde{O}$(vrank) iterations and faster algorithms for maximum flow. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 424–433, 2014.

44 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs on bounded treewidth are probably optimal. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 777–789, 2011.

45 J. Erickson M. Soss and M. H. Overmars. Preprocessing chains for fast dihedral rotations is hard or even impossible. *Computational Geometry: Theory and Applications*, 26(3):235–246, 2002.

46 Aleksander Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 253–262, 2013.

47 William J Masek and Michael S Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System sciences*, 20(1):18–31, 1980.

48 B. Monien and E. Speckenmeyer. Solving satisfiability in less than 2^n steps. *Discrete Applied Mathematics*, 10(3):287–295, 1985.

49 J. Nešetřil and S. Poljak. On the complexity of the subgraph problem. *Commentationes Math. Universitatis Carolinae*, 26(2):415–419, 1985.

50 R. Paturi, P. Pudlák, M. E. Saks, and F. Zane. An improved exponential-time algorithm for k-SAT. *J. ACM*, 52(3):337–364, 2005.

51 R. Paturi, P. Pudlák, and F. Zane. Satisfiability coding lemma. *Chicago J. Theor. Comput. Sci.*, 1999, 1999.

52 Marcin Pilipczuk and Michał Pilipczuk. Finding a maximum induced degenerate subgraph faster than 2^n. In *IPEC'12: Parameterized and Exact Computation*, pages 3–12, 2012.

53 Mihai Pătraşcu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 603–610, 2010.

54 L. Roditty and V. Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing*, STOC'13, pages 515–524, New York, NY, USA, 2013. ACM.

55 Liam Roditty and Uri Zwick. On dynamic shortest paths problems. In *Algorithms – ESA 2004, 12th Annual European Symposium, Bergen, Norway, September 14-17, 2004, Proceedings*, pages 580–591, 2004.

56 Ingo Schiermeyer. Solving 3-satisfiability in less than 1.579^n steps. In *Computer Science Logic, 6th Workshop, CSL'92, San Miniato, Italy, September 28 – October 2, 1992, Selected Papers*, pages 379–394, 1992.

57 Uwe Schöning. A probabilistic algorithm for k-sat and constraint satisfaction problems. In *40th Annual Symposium on Foundations of Computer Science, FOCS'99, 17-18 October, 1999, New York, NY, USA*, pages 410–414, 1999.

58 Virginia Vassilevska and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 – June 2, 2009*, pages 455–464, 2009.

59 V. Vassilevska Williams and R. Williams. Finding, minimizing, and counting weighted subgraphs. *SIAM J. Comput.*, 42(3):831–854, 2013.

60 Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19–22, 2012*, pages 887–898, 2012.

61 Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 645–654, 2010.

62 R. Williams. Personal communication, 2015.

63 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005.

64 Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 – June 03, 2014*, pages 664–673, 2014.

65 Ryan Williams and Huacheng Yu. Finding orthogonal vectors in discrete structures. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1867–1877, 2014.

Variants of Plane Diameter Completion*

Petr A. Golovach[1], Clément Requilé[2], and Dimitrios M. Thilikos[3,4,5]

1 **Department of Informatics, University of Bergen, Bergen, Norway**
Petr.Golovach@ii.uib.no
2 **Freie Universität Berlin, Institut für Mathematik und Informatik, Berlin, Germany**
requile@math.fu-berlin.de
3 **AlGCo project-team, CNRS, LIRMM, Montpellier, France**
sedthilk@thilikos.info
4 **Department of Mathematics, University of Athens, Athens, Greece**
5 **Computer Technology Institute & Press "Diophantus", Patras, Greece**

Abstract

The Plane Diameter Completion problem asks, given a plane graph G and a positive integer d, if it is a spanning subgraph of a plane graph H that has diameter at most d. We examine two variants of this problem where the input comes with another parameter k. In the first variant, called BPDC, k upper bounds the total number of edges to be added and in the second, called BFPDC, k upper bounds the number of additional edges per face. We prove that both problems are NP-complete, the first even for 3-connected graphs of face-degree at most 4 and the second even when $k = 1$ on 3-connected graphs of face-degree at most 5. In this paper we give parameterized algorithms for both problems that run in $O(n^3) + 2^{2^{O((kd)^2 \log d)}} \cdot n$ steps.

1998 ACM Subject Classification G.2.1 Combinatorics, G.2.2 Graph Theory

Keywords and phrases Planar graphs, graph modification problems, parameterized algorithms, dynamic programming, branchwidth

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.30

1 Introduction

In 1987, Chung [1, Problem 5] introduced the following problem[1]: find the optimum way to add q edges to a given graph G so that the resulting graph has minimum diameter. This problem was proved to be NP-hard if the aim is to obtain a graph of diameter at most 3 [14], and later the NP-hardness was shown even for the Diameter-2 Completion problem [9]. It is also know that Diameter-2 Completion is W[2]-hard when parameterized by q [6].

For planar graphs, Dejter and Fellows introduced in [3] the Planar Diameter Completion problem that asks whether it is possible to obtain a planar graph of diameter at most

* The first author was supported by the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 267959. The second author was supported by the FP7-PEOPLE-2013-CIG project CountGraph (ref. 630749), the collateral PROCOPE-DAAD project RanConGraph (ref. 57134837), and the Berlin Mathematical School. The research of the third author was co-financed by the European Union (European Social Fund ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF), Research Funding Program: ARISTEIA II.

[1] Notice that in all problems defined in this paper we can directly assume that G is a simple graph as loops do not contribute to the diameter of a graph and the same holds if we take simple edges instead of multiple ones.

d from a given planar graph by edge additions. It is not known whether Planar Diameter Completion admits a polynomial time algorithm, but Dejter and Fellows showed that, when parameterized by d, Planar Diameter Completion is fixed parameter tractable [3]. The proof is based on the fact that the yes-instances of the problem are closed under taking minors. Because of the Robertson and Seymour theorem [13] and the algorithm in [11], this implies that, for each d, the set of graphs G for which (G, d) is a yes-instance can be characterized by a *finite* set of forbidden minors. This fact, along with the minor-checking algorithm in [12] implies that there exists an $O(f(d) \cdot n^3)$-step algorithm (i.e. an FPT-algorithm) deciding whether a plane graph G has a plane completion of diameter at most d. Using the parameterized complexity, this means that Planar Diameter Completion is FPT, when parameterized by d. To make this result constructive, one requires the set of forbidden minors for each d, which is unknown. To find a constructive FPT-algorithm for this parameterized problem remains a major open problem in parameterized algorithm design.

Our results. We denote by $\mathbb{S}_0$ the 3-dimensional sphere. By a *plane* graph G we mean a simple planar graph G with the vertex set $V(G)$ and the edge set $E(G)$ drawn in $\mathbb{S}_0$ such that no two edges of this embedding intersect. A plane graph H is a *a plane completion* (or, simply *completion*) of another plane graph G if H is a spanning subgraph of G. A *q-edge completion* of a plane graph G is a completion H of G where $|E(H)| - |E(G)| \leq q$. A *k-face completion* of a plane graph G is a completion H of G where at most k edges are added in each face of G. We consider the following problem:

Plane Diameter Completion (PDC)
Input: a plane graph G and $d \in \mathbb{N}_{\geq 1}$.
Output: is there a completion of G with diameter at most d?

An important difference between PDC and the aforementioned problems is that we consider plane graphs, i.e., the aim is to reduce the diameter of a given embedding of a planar graph preserving the embedding. In particular, we are interested in the following variants:

Bounded Budget PDC (BPDC)
Input: a plane graph G and $q \in \mathbb{N}, d \in \mathbb{N}_{\geq 1}$
Question: is there a completion H of G of diameter at most d that is also a q-edge completion?

Bounded Budget/Face PDC (BFPDC)
Input: a plane graph G and $k \in \mathbb{N}, d \in \mathbb{N}_{\geq 1}$.
Question: is there a completion H of G of diameter at most d that is also a k-face completion?

We examine the complexity of the two above problems. Our hardness results are the following.

▶ **Theorem 1.** *Both* BPDC *and* BFPDC *are* NP*-complete. Moreover,* BPDC *is* NP*-complete even for 3-connected graphs of face-degree at most 4, and* BFPDC *is* NP*-complete even for $k = 1$ on 3-connected graphs of face-degree at most 5.*

The hardness results are proved using a series of reductions departing from the Planar 3-Satisfiability problem that was shown to be NP-hard by Lichtenstein in [10].

The results of Theorem 1 prompt us to examine the parameterized complexity of the above problems (for more on parameterized complexity, we refer the reader to [5]). For this, we consider the following general problem:

Bounded Budget and Budget/Face BDC (BBFPDC)
Input: a plane graph G, $q \in \mathbb{N} \cup \{\infty\}$, $k \in \mathbb{N}$, and $d \in \mathbb{N}_{\geq 1}$.
Question: is there a completion H of G of diameter at most d that is also a q-edge completion and a k-face completion?

Notice that when $q = \infty$ BBFPDC yields BFPDC and when $q = k$ BBFPDC yields BPDC. Our main result is that BBFPDC is fixed parameter tractable (belongs in the parameterized class FPT) when parameterized by k and d.

▶ **Theorem 2.** *It is possible to construct an* $O(n^3) + 2^{2^{O((kd)\log d)}} \cdot (\alpha(q))^2 \cdot n$*-step algorithm for* BBFPDC*.*

In the above statement and in the rest of this paper we use the function $\alpha : \mathbb{N} \cup \{\infty\} \to \mathbb{N}$ such that if $q = \infty$, then $\alpha(q) = 1$, otherwise $\alpha(q) = q$.

The main ideas of the algorithm of Theorem 2 are the following. We first observe that YES-instances of PDC and all its variants have bounded branchwidth (for the definition of branchwidth, see Section 2). The typical approach in this case is to derive an FPT-algorithm by either expressing the problem in Monadic Second Order Logic – MSOL (using Courcelle's theorem [2]) or to design a dynamic programming algorithm for this problem. However, for completion problems, this is not really plausible as this logic can quantify on *existing* edges or vertices of the graph and not on the "non-existing" completion edges. This also indicates that to design a dynamic programming algorithm for such problems is, in general, not an easy task. In this paper we show how to tackle this problem for BBFPDC (and its special cases BPDC and BFPDC). Our approach is to deal with the input G as a part of a more complicated graph with $O(k^2 \cdot n)$ additional edges, namely its *cylindrical enhancement* G' (see Section 3 for the definition). Informally, sufficiently large cylindrical grids are placed inside the faces of G and then internally vertex disjoint paths in these grids can be used to emulate the edges of a solution of the original problem placed inside the corresponding faces. Thus, by the enhancement we reduce BBFPDC to a new problem on G' certified by a suitable 3-partition of the additional edges. Roughly, this partition consists of the 1-weighted edges that should be added in the completion, the 0-weighted edges that should link these edges to the boundary of the face of G where they will be inserted, and the ∞-weighted edges that will be the (useless) rest of the additional edges. The new problem asks for such a partition that simulates a bounded diameter completion. The good news is that, as long as the number of edges per face to be added is bounded, which is the case for BBFPDC, the new graph G' has still bounded branchwidth and it is possible, in the new instance, to quantify this 3-partition of the graph G'. However, even under these circumstances, to express the new problem in Monadic Second Order Logic is not easy. For these reasons we decided to follow the more technical approach of designing a dynamic programming algorithm that leads to the (better) complexity bounds of Theorem 2. This algorithm is quite involved due to the technicalities of the translation of the BBFPDC to the new problem. It runs on a sphere-cut decomposition of the plane embedding of G' and its tables encode how a partial solution is behaving inside a closed disk whose boundary meets only (a few of) the edges of G'. We stress that this encoding takes into account the topological embedding and not just the combinatorial structure of G'. Sphere-cut decompositions as well as some necessary combinatorial structures for this encoding are presented in Section 4. The dynamic programming algorithms is presented in Section 5 and is the most technical part of this paper.

Due to space restrictions, various proofs are omitted in this extended abstract and are available in [7].

2 Definitions and preliminaries

Given a graph G, we denote by $V(G)$ (respectively $E(G)$) the *set of vertices* (respectively *edges*) of G. A graph G' is a *subgraph* of a graph G if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$, and we denote this by $G' \subseteq G$. Also, in case $V(G) = V(G')$, we say that H is a *spanning subgraph* of G. If S is a set of vertices or a set of edges of a graph G, the graph $G \setminus S$ is the graph obtained from G after the removal of the elements of S. If S is a set of edges, we define $G[E]$ as the graph whose vertex set consists of the endpoints of the edges of E and whose edge set of E.

Distance and diameter. Let G be a graph and let $\mathsf{w} : E(G) \to \mathbb{N} \cup \{\infty\}$ (w is a *weighting of the edges of* G). Given two vertices $x, x' \in V(G)$ we call (x, x')-*path* every path of G with x and x' as endpoints. We also define $\mathsf{w}\text{-}\mathbf{dist}_G(x, x') = \min\{\mathsf{w}(E(P)) \mid P \text{ is an } (x, x')\text{-path in } G\}$.

Plane graphs. To simplify notations on plane graphs, we consider a plane graph G as the union of the points of $\mathbb{S}_0$ in its embedding corresponding to its vertices and edges. That way, a subgraph H of G can be seen as a graph H where $H \subseteq G$. The *faces* of a plane graph G, are the connected components of the set $\mathbb{S}_0 \setminus G$. A vertex v (an edge e resp.) of a plane graph G is *incident* to a face f and, vice-versa, f is incident to v (resp. e) if v (resp., e) lies on the boundary of f. The *degree* of a face f of G is the number of edges incident to f where bridges of G count double in this number. The *face-degree* of G is the maximum degree of a face in $F(G)$. A set $\Delta \subseteq \mathbb{S}_0$ is an open disc if it is homeomorphic to $\{(x, y) : x^2 + y^2 < 1\}$. Also, Δ is a *closed disk* of $\mathbb{S}_0$ if it is the closure of some open disk of $\mathbb{S}_0$.

Branch decomposition. Given a graph H with n vertices, a branch decomposition of H is a pair (T, μ), where T is a tree with all internal vertices of degree three and $\mu : L \to E(H)$ is a bijection from the set of leaves of T to the edges of H. For every edge e of T, we define the middle set $\mathbf{mid}(e) \subseteq V(H)$ as follows: if $T \setminus \{e\}$ has two connected components T_1 and T_2, and for $i \in \{1, 2\}$, let $H_i^e = H[\{\mu(f) : f \in L \cap V(T_i)]$, and set $\mathbf{mid}(e) = V(H_1^e) \cap V(H_2^e)$. The width of (T, μ) is the maximum order of the middle sets over all edges of T, i.e. $\max\{|\mathbf{mid}(e)| : e \in T\}$. The *branchwidth* of H is the minimum width of a branch decomposition of H and is denoted by $\mathbf{bw}(H)$.

We use the following lemma.

▶ **Lemma 3.** *There exists a constant c_1 such that if (G, d) is a* YES*-instance of* PDC*, then $\mathbf{bw}(G) \leq c_1 \cdot d$. The same holds for the graphs in the* YES*-instances of* BPDC*,* BFPDC*, and* BBFPDC*.*

3 The reduction

Edge colorings of new edges. Let G and H be two plane graphs such that G is a subgraph of H and let $q \in \mathbb{N} \cup \{\infty\}$, $k \in \mathbb{N}$, and $d \in \mathbb{N}_{\geq 1}$. Given a 3-partition $\mathbf{p} = \{E^0, E^1, E^\infty\}$ of $E(H) \setminus E(G)$, we define the function $\mathsf{w}_\mathbf{p} : E(H) \to \mathbb{N}$ such that

$$\mathsf{w}_\mathbf{p} = \{(e, 1) \mid e \in E(G)\} \cup \{(e, 0) \mid E \in E^0\} \cup \{(e, 1) \mid e \in E^1\} \cup \{(e, d+1) \mid E \in E^\infty\}.$$

We say that G has (q, k, d)-*extension* in H if there is a 3-partition $\mathbf{p} = \{E^0, E^1, E^\infty\}$ of $E(H) \setminus E(G)$ such that the following conditions hold

A. There is no path in H with endpoints in $V(G)$ that consists of edges in E^0,

B. every face F of G contains at most k edges of E^1,

C. $\forall x, y \in V(G), \mathsf{w_p\text{-}dist}_H(x, y) \leq d$, and
D. $|E^1| \leq q$.

Given a 3-partition $\mathbf{p} = \{E^0, E^1, E^\infty\}$ of $E(H) \setminus E(G)$ we refer to its elements as the *0-edges*, the *1-edges*, and the *∞-edges* respectively. We also call the edges of G *old-edges*.

▶ **Lemma 4.** *There exists a $c_2 \in \mathbb{Z}_{\geq 1}$ and an algorithm that receives as input a planar graph G on n vertices and a positive integer k and outputs a 3-connected planar graph G_w where*

- $\mathbf{bw}(G_k) \leq c_2 \cdot k \cdot \mathbf{bw}(G))$.
- *For every $q \in \mathbb{N} \cup \{\infty\}$ and $d \in \mathbb{N}_{\geq 1}$, (G, q, k, d) is a* YES*-instance of* BBFPDC *if and only if G has a (q, k, d)*-extension *in G_k.*

Moreover, this algorithm runs in $O(k^2 \cdot n)$ steps.

4 Structures for dynamic programming

For our dynamic programming algorithm we need a variant of branchwidth for plane graphs whose middle sets have additional topological properties.

Sphere-cut decomposition. Let H be a plane graph. An arc is a subset O of the plane homeomorphic to a circle and is called a *noose of H* if it meets H only in vertices. We also set $V_O = V(H) \cap O$. An *arc* of a noose O is a connected component of $O \setminus V_O$ while in the trivial case where $V_O = \emptyset$, O does not have arcs. A *sphere-cut decomposition* or *sc-decomposition* of H is a triple (T, μ, π) where (T, μ) is a branch decomposition of H and π is a function mapping each $e \in E(T)$ to cyclic orderings of vertices of H, such that for every $e \in E(T)$ there is a noose O_e of H where the following properties are satisfied:

- O_e meets every face of H at most once,
- H_1^e is contained in one of the closed disks bounded by O_e and H_2^e is contained in the other (H_1^e and H_2^e are as in the definition of branch decomposition).
- $\pi(e)$ is a cyclic ordering of V_{O_e} defined by a clockwise traversal of O_e in the embedding of H.

We denote $X_e = V_{O_e}$ and we always assume that its vertices are clockwise enumerated according to $\pi(e)$. We denote by $\mathbf{A}_e$ the set containing the arcs of O_e. Also, if $\pi(e) = [a_1, \ldots, a_k, a_1]$, then we use the notation $\mathbf{A}_e = \{a_{1,2}, a_{2,3}, \ldots, a_{k-1,k}, a_{k,1}\}$ where the boundary of the arc $a_{i,i+1}$ consists of the vertices a_i and a_{i+1}. We also define $H_e^+ = (V(H), E(H \cup \mathbf{A}_e))$, i.e., H_e^+ is the embedding occurring if we add in H the arcs of O_e as edges. A face of H_e^+ is called *internal* if it is not incident to an arc in $\mathbf{A}_e$, i.e., it is also a face of H. A face of H_e^+ is *marginal* if it is a properly included is some face of H.

For our dynamic programming we require to have in hand an optimal sphere-cut decomposition. This is done combining the main result of [8] and [15, (5.1)] (see also [4]) and is summarized to the following.

▶ **Proposition 5.** *There exists an algorithm that, with input a 3-connected plane graph G and $w \in \mathbb{N}$, outputs a sphere-cut decomposition of G of width at most w or reports that $\mathbf{bw}(G) > w$.*

Our next step is to define a series of combinatorial structures that are necessary for our dynamic programming. Given two sets A and B we denote by A^B the set of all functions from B to A.

(d, k, q)-configurations. Given a set X and a non-negative integer t, we say that the pair $(\mathcal{X}, \chi)$ is a *t-labeled partition of X* if $\mathcal{X}$ is a collection of pairwise disjoint non-empty subsets of X and χ is a function mapping the integers in $\{1, \ldots, |\mathcal{X}|\}$ to integers in $\{0, \ldots, t\}$. In case $X = \emptyset$, a *t-labeled partition* corresponds to the pair $\{\emptyset, \varnothing\}$ where $\varnothing$ is the "empty" function, i.e. the function whose domain is empty. Let X and A be two finite sets. Given $d, k \in \mathbb{N}$ and $q \in \mathbb{N} \cup \{\infty\}$, we define a *$(d, k, q)$-configuration* of (X, A) as a quintuple $((\mathcal{X}, \chi), (\mathcal{A}, \alpha), (\mathcal{F}, \mathcal{E}), \delta, z)$ where

1. $(\mathcal{X}, \chi)$ is a 1-labeled partition of X,
2. $(\mathcal{A}, \alpha)$ is a k-labeled partition of A,
3. $(\mathcal{F}, \mathcal{E})$ is a graph (possibly with loops) where $\mathcal{F} \subseteq \{0, \ldots, d+1\}^X$,
4. $\delta \in \{0, \ldots, d+1\}^{X^2}$, and
5. if $q \in \mathbb{N}$, then $z \leq q$, otherwise $z = \infty$.

Fusions and restrictions. Let $(\mathcal{X}_1, \chi_1)$ and $(\mathcal{X}_2, \chi_2)$ be two t-labeled partitions of the sets X_1 and X_2 respectively such that $\mathcal{X}_i = \{X_1^i, \ldots, X_{\rho_1}^i\}, i \in \{1, 2\}$. We define $\mathcal{X}_1 \oplus \mathcal{X}_2$ as follows: if $x, x' \in X_1 \cup X_2$ we say that $x \sim x'$ if there is a set in $\mathcal{X}_1 \cup \mathcal{X}_2$ that contains both of them. Let $\sim_T$ be the transitive closure of $\sim$. Then $\mathcal{X}_1 \oplus \mathcal{X}_2$ contains the equivalence classes of $\sim_T$. We now define $\chi_1 \oplus \chi_2$ as follows: let $\mathcal{X}_1 \oplus \mathcal{X}_2 = \{Y_1, \ldots, Y_\rho\}$. Then for each $i \in \{1, \ldots, \rho\}$, we define $\chi_1 \oplus \chi_2(i) = \min\{t, \sum_{X_{i'}^1 \subseteq Y_i} \chi_1(i') + \sum_{X_{i'}^2 \subseteq Y_i} \chi_2(i')\}$.

The *fusion* of the t-labeled partitions $(\mathcal{X}_1, \chi_1)$ and $(\mathcal{X}_2, \chi_2)$ is the pair $(\mathcal{X}_1 \oplus \mathcal{X}_2, \chi_1 \oplus \chi_2)$ that is a $(t+1)$-labeled partition and is denoted by $(\mathcal{X}_1, \chi_1) \oplus (\mathcal{X}_2, \chi_2)$. Given a t-labeled partition $(\mathcal{X}, \chi)$ of a set X and given a subset X' of X we define the *restriction* of $(\mathcal{X}, \chi)$ to X' as the t-labeled partition $(\mathcal{X}', \chi')$ of X' where $\mathcal{X}' = \{X_i \cap X' \mid X_i \in \mathcal{X}\} \setminus \{\emptyset\}$ and $\chi' = \{(i, \chi(i)) \mid X_i \cap X' \neq \emptyset\}$ and we denote it by $(\mathcal{X}, \chi)|_{X'}$. We also define the intersection of $(\mathcal{X}, \chi)$ with X' as the t-labeled partition $(\mathcal{X}', \chi')$ where $\mathcal{X}' = \{X_i \in \mathcal{X} \mid X_i \cap (X \setminus X') \neq \emptyset\}$ and $\chi' = \{(i, \chi(i)) \mid X_i \cap X'' \neq \emptyset\}$ where $X'' = \cup_{X_i' \in \mathcal{X}'} X_i$ and we denote it by $(\mathcal{X}, \chi) \cap X'$. Notice that $(\mathcal{X}, \chi)|_{X'}$ and $(\mathcal{X}, \chi) \cap X'$ are not always the same.

5 Dynamic programming

The following result is the main algorithmic contribution of this paper.

▶ **Lemma 6.** *There exists an algorithm that, given (G, H, q, k, d, D, b) as input where G and H are plane graphs such that G is a subgraph of H, H is 3-connected, $q \in \mathbb{N} \cup \{\infty\}, k \in \mathbb{N}$, $d \in \mathbb{N}_{\geq 1}$, $b \in \mathbb{N}$, and $D = (T, \mu, \pi)$ is a sphere-cut decomposition of H with width at most b, decides whether G has* (q, k, d)-extension *in H in $(\alpha(q))^2 \cdot 2^{O(b^2 \log d) + 2^{O(b \log d)}} \cdot n$ steps.*

Proof. We use the notation $E^{\text{old}} = E(G)$ and $E^{\text{new}} = E(H) \setminus E(G)$, $V^{\text{old}} = V(G)$ and $V^{\text{new}} = V(H) \setminus V(G)$. We choose an arbitrary edge $e^* \in E(T)$, subdivide it by adding a new vertex v_{new} and update T by adding a new vertex r adjacent to v_{new}. We then root T at this vertex r and we extend μ by setting $\mu(r) = \emptyset$. In T we call *leaf-edges* all its edges that are incident to its leaves except from the edge $e_r = \{r, v_{\text{new}}\}$. An edge of T that is not a leaf-edge is called *internal*. We denote by $L(T)$ the set of the leaf-edges of T and we denote by $I(T)$ the internal edges of T. We also call e_r *root-edge*. For each $e \in E(T)$, let T_e be the tree of the forest $T \setminus \{e\}$ that does not contain r as a leaf and let E_e be the edges that are images, via μ, of the leaves of T that are also leaves of T_e. We denote $H_e = H[E_e]$ and $V_e = V(H_e)$ and observe that $H_{e_r} = H$. For each edge $e \in I(T)$, we define its children as the two edges that both belong in the connected component of $T \setminus e$ that does not contain the root r and that share a common endpoint with e. Also, for each edge $e \in E(T)$, we

define Δ_e as the closed disk bounded by O_e such that $G \cap \Delta_e = H_e$. Finally, for each edge $e \in E(T)$, we set $X_e = \mathbf{mid}(e)$, $V_e^{\text{new}} = V_e \cap V^{\text{new}}$, $V_e^{\text{old}} = V_e \cap V^{\text{old}}$, $E_e^{\text{new}} = E_e \cap E^{\text{new}}$, and $E_e^{\text{old}} = E_e \cap E^{\text{old}}$.

Distance signatures and dependency graphs. Let $\mathbf{p} = \{E_e^0, E_e^1, E_e^\infty\}$ be a 3-partition of E_e^{new}. For each vertex $v \in V_e$, we define the $(X_e, \mathbf{p})$-*distance vector* of v as the function $\phi_v : X_e \to \{0, \ldots, d+1\}$ such that if $x \in X_e$ then $\phi_v(x) = \min\{\mathsf{w_p}\text{-}\mathbf{dist}_{G_e}(v, x), d+1\}$. We define the $(e, \mathbf{p})$-*dependency graph* $\mathcal{G}_{e,\mathbf{p}} = (\mathcal{F}_{e,\mathbf{p}}, \mathcal{E}_{e,\mathbf{p}})$ (that may contain loops) where $\mathcal{F}_{e,\mathbf{p}} = \{\phi_v \mid v \in V_e\}$ and such that two (not necessarily distinct) vertices ϕ and ϕ' of $\mathcal{F}_{e,\mathbf{p}}$ are connected by an edge in $\mathcal{E}_{e,\mathbf{p}}$ if and only if there exist $v, v' \in V_e$ such that $\phi = \phi_v$, $\phi' = \phi_{v'}$ and $\mathsf{w_p}\text{-}\mathbf{dist}_{H_e}(v, v') > d$. Notice that the set $\Phi_e = \{\mathcal{G}_{e,\mathbf{p}} \mid \mathbf{p} \text{ is a 3-partition of } E_e^{\text{new}}\}$ has at most $2^{(d+2)^{|X_e|}}$ elements because $\{\mathcal{F}_{e,\mathbf{p}} \mid \mathbf{p} \text{ is a 3-partition of } E_e^{\text{new}}\} \subseteq \{0, \ldots, d+1\}^{X_e}$ and, to each $\mathcal{F}_{e,\mathbf{p}}$, assign a unique edge set $\mathcal{E}_{e,\mathbf{p}}$. Intuitively, each $\mathcal{F}_{e,\mathbf{p}}$ corresponds to a partition of the elements of V_e such that vertices in the same part have the same $(X_e, \mathbf{p})$-distance signature. Moreover the existence of an edge in the $(e, \mathbf{p})$-dependency graph between two such parts implies that they contain vertices, one from each part, whose $\mathsf{w_p}$-distance in H_e is bigger than d.

The tables. Our aim is to give a dynamic programming algorithm running on the sc-decomposition T. For this, we describe, for each $e \in E(T)$, a table $\mathfrak{T}(e)$ containing information on partial solutions of the problem for the graph G_e in a way that the table of an edge $e \in E(T)$ can be computed using the tables of the two children of e, the size of each table does not depend on G and the final answer can be derived by the table of the root-edge e_r.

We define the function $\mathfrak{T}$ mapping each $e \in E(T)$ to a collection $\mathfrak{T}(e)$ of (d, k, q)-configurations of $(X_e, \mathbf{A}_e)$. In particular, $Q = ((\mathcal{X}, \chi), (\mathcal{A}, \alpha), (\mathcal{F}, \mathcal{E}), \delta, z) \in \mathfrak{T}(e)$ iff there exists a 3-partition $\mathbf{p} = \{E_e^0, E_e^1, E_e^\infty\}$ of E_e^{new} such that the following hold:

1. $C_1, \ldots, C_h$ are the connected components of $(V(H_e), E_e^0)$, then
   - $\mathcal{X} = \{V(C_1) \cap X_e, \ldots, V(C_h) \cap X_e\}$ and
   - $\forall_{i \in \{1,\ldots,h\}}$ $\chi(i) = 1$ if C_i contains some vertex of V_e^{old}, otherwise $\chi(i) = 0$.

   (The pair $(\mathcal{X}, \chi)$ encodes the connected components of the 0-edges that contain vertices of X_e and for each of them registers the number (0 or 1) of the vertices in V_e^{old} in them. This information is important to control Condition A.)
2. $\mathcal{A}$ is a partition of $\mathbf{A}_e$ such that two arcs $A, A' \in \mathbf{A}_e$ belong in the same set, say A_i of $\mathcal{A}$ if and only if they are incident to the same marginal face f_i of H_e^+. Moreover, for each $i \in \{1, \ldots, |\mathcal{A}|\}$, $\alpha(i)$ is equal to the number of edges in E_e^1 that are inside f_i.
   (Here $(\mathcal{A}, \alpha)$ encodes the "partial" faces of the embedding of G_e that are inside Δ_e. To each of them we correspond the number of 1-edges that they contain in H_e. This is useful in order to guarantee that during the algorithm, faces that stop being marginal do not contain more than k 1-edges, as required by Condition B.)
3. $(\mathcal{F}, \mathcal{E})$ is the $(e, \mathbf{p})$-dependency graph, i.e., the graph $\mathcal{G}_{e,\mathbf{p}} = (\mathcal{F}_{e,\mathbf{p}}, \mathcal{E}_{e,\mathbf{p}})$.
   (Recall that $\mathcal{F}$ is the collection of all the different distance vectors of the vertices of V_e. Notice also that there might be pairs of vertices $x, x' \in V_e$ whose $\mathbf{w_p}$-distance in G_e is bigger than d. In order for G to have a completion of diameter d, these two vertices should become connected, at some step of the algorithm, by paths passing *outside* Δ_e. To check this possibility, it is enough to know the distance vectors of x and x' and these are encoded in the set $\mathcal{F}$. Moreover the fact that x and x' are still "far away" inside G_e is certified by the existence of an edge (or a loop) between their distance vectors in $\mathcal{F}$.)

4. For each pair $x, x' \in X_e$, $\delta(x, x') = \min\{\mathsf{w_p}\text{-}\mathbf{dist}_{H_e}(x, x'), d+1\}$.
   (This information is complementary to the one stored in $\mathcal{F}$ and registers the distances of the vertices in X_e inside H_e. As we will see, $\mathcal{F}$ and δ will be used in order to compute the distance vectors as well as their dependencies during the steps of the algorithm.)
5. There is no path in H_e with endpoints in V_e^{old} that consists of edges in E_e^0.
   (This ensures that Condition A is satisfied for the current graph G_e.)
6. Every internal face of G_e^+ contains at most k edges in E_e^1.
   (This ensures that Condition B holds for all the internal faces of G_e.)
7. $\forall v, v' \in V_e$, either $\mathsf{w_p}\text{-}\mathbf{dist}_{H_e}(v, v') \leq d$ or there are two vertices $x, x' \in X_e$ such that $\phi_v(x) + \phi_{v'}(x') \leq d$.
   (Here we demand that if two vertices x_1, x_2 of V_e are "far away" (have $\mathbf{w_p}$-distance $> d$) inside H_e then they have some chance to come "close" (obtain $\mathbf{w_p}$-distance $\leq d$) in the final graph, so that Condition C is satisfied. This fact is already stored by an edge in $\mathcal{E}$ between the two distance vectors of x and x' and the possibility that x_1 and x_2 may come close at some step of the algorithm, in what concerns the graph G_e, depends only on these distance vectors and not on the vertices x_1 and x_2 themselves.)
8. There are at most z edges of E_e^1 inside the internal faces of G_e^+ (clearly, this last condition becomes void when $q = \infty$).
   (This information helps us control Condition D during the algorithm.)

Notice that in case $X_e = \emptyset$ the only graph that can correspond to the 6th step is the graph $(\{\varnothing\}, \emptyset)$ which, from now on will be denoted by $G_\varnothing$.

Bounding the set of characteristics. Our next step is to bound $\mathfrak{T}(e)$ for each $e \in E(T)$. Notice first that $|X_e| = |\mathbf{A}_e| \leq b$. This means that there are $2^{O(b \log b)}$ instantiations of $(\mathcal{X}, \chi)$ and $2^{O(k+b \log b)}$ instantiations of $(\mathcal{A}, \alpha)$. As we previously noticed, the different instantiations of $(\mathcal{F}, \mathcal{E})$ are $|\Phi_e| = 2^{2^{O(b \log d)}}$. Moreover, there are $2^{O(b^2 \log d)}$ instantiations of δ and $\alpha(q)$ instantiations of z. We conclude that there exists a function f such that for each $e \in V(T)$, $|\mathfrak{T}(e)| \leq f(k, q, b, d)$. Moreover, $f(k, q, b, d) = \alpha(q) \cdot 2^{O(b^2 \log d) + 2^{O(b \log d)}}$.

The characteristic function on the root edge. Observe that E_{new} is (k, d, q, w)-edge colorable in H if and only if $\mathfrak{T}(e_r) \neq \emptyset$, i.e., $((\emptyset, \varnothing), (\emptyset, \varnothing), G_\varnothing, \varnothing, z) \in \mathfrak{T}(e_r)$ for some $z \leq q$. Indeed, if this happens, conditions 1–4 become void while conditions 5, 6, 7, and 8 imply that $H = H_e$ satisfies the conditions A, B, C, and D respectively in the definition of the (k, d, q, w)-edge colorability of E^{new}.

The computation of the tables. We will now show how to compute $\mathfrak{T}(e)$ for each $e \in E(T)$.

We now give the definition of $\mathfrak{T}(e)$ in the case where e is a leaf of T is the following: Given a $q \in \mathbb{N} \cup \{\infty\}$, we define $A(q) = \{\infty\}$ if $q = \infty$, otherwise $A(q) = \{z \mid z \leq q\}$.

Suppose now that e_l is a leaf-edge of T where $\pi(e_l) = [a_1, a_2, a_1]$ and $\mathbf{A}_{e_l} = \{a_{1,2}, a_{2,1}\}$.

1. If $\{a_1, a_2\} \in E_e^{\text{old}}$, then

$$\begin{aligned}\mathfrak{T}(e_l) \quad = \quad \{\ & \big((\{\{a_1\}, \{a_2\}\}, \{(1,1), (2,1)\}),\\ & (\{\{a_{1,2}\}, \{a_{2,1}\}\}, \{(1,0), (2,0)\}),\\ & \big(\big\{\{(a_1, 0), (a_2, \mathsf{w}(\{a_1, a_2\}))\}, \{(a_1, \mathsf{w}(\{a_1, a_2\})), (a_2, 0)\}\big\}, \emptyset\big),\\ & \{((a_1, a_2), \mathsf{w}(\{a_1, a_2\}))\}, z\big) \mid z \in A(q)\},\end{aligned}$$

2. if $\{a_1, a_2\} \in E_e^{\text{new}}$ and $\{a_1, a_2\} \subseteq V_e^{\text{old}}$, then $\mathfrak{T}(e_l) = \mathcal{Q}^1 \cup \mathcal{Q}^\infty$ where

$$\begin{aligned}
\mathcal{Q}^1 \;=\; \{\; & \big(\,(\{\{a_1\},\{a_2\}\},\{(1,1),(2,1)\}) \\
& (\{\{a_{1,2},a_{2,1}\}\},\{(1,1)\}) \\
& (\big\{\{(a_1,0),(a_2,1)\},\{(a_1,1),(a_2,0)\}\big\},\emptyset) \\
& \{((a_1,a_2),s)\},z\,\big) \mid z \in A(q) - \{0\}\} \\
\mathcal{Q}^\infty \;=\; \{\; & \big(\,(\{\{a_1\},\{a_2\}\},\{(1,1),(2,1)\}) \\
& (\{\{a_{1,2},a_{2,1}\}\},\{(1,0)\}) \\
& (\big\{\{(a_1,0),(a_2,d+1)\},\{(a_1,d+1),(a_2,0)\}\big\},K) \\
& \{((a_1,a_2),d+1)\},z\,\big) \mid z \in A(q)\}
\end{aligned}$$

(the set K above contains a single edge that is not a loop), and if $\{a_1, a_2\} \in E_e^{\text{new}}$ and $\{a_1, a_2\} \nsubseteq V_e^{\text{old}}$, then $\mathfrak{T}(e_l) = \mathcal{Q}^1 \cup \mathcal{Q}^\infty \cup \mathcal{Q}^0$ where

$$\begin{aligned}
\mathcal{Q}^0 \;=\; \{\; & \big(\,(\{\{a_1,a_2\}\},\{(1,1-\langle\{a_1,a_2\} \subseteq V_e^{\text{new}}\rangle)\}) \\
& (\{\{a_{1,2},a_{2,1}\}\},\{(1,0)\}) \\
& (\{\{(a_1,0),(a_2,0)\}\},\emptyset) \\
& \{((a_1,a_2),0)\},z\,\big) \mid z \in A(q)\}.
\end{aligned}$$

Assume now that e is a non-leaf edge of T with children e_l and e_r, the collection $\mathfrak{T}(e)$ is given by $\mathbf{join}(\mathfrak{T}(e_1), \mathfrak{T}(e_2))$ where **join** is a procedure that is depicted below. Notice that $\mathbf{A}_e$ is the symmetric difference of $\mathbf{A}_{e_l}$ and $\mathbf{A}_{e_r}$ and X_e consists of the endpoints of the arcs in $\mathcal{A}_e$. We also set $X_e^F = (X_{e_l} \cup X_{e_r}) \setminus X_e$.

Procedure **join**
Input: two collections $\mathcal{C}_{e_l}$ and $\mathcal{C}_{e_r}$ of (d,k,q)-configurations of $(X_{e_l}, \mathbf{A}_{e_l})$ and $(X_{e_r}, \mathbf{A}_{e_r})$.
Output: a collection $\mathcal{C}_r$ of (d,k,q)-configurations of $(X_e, \mathbf{A}_e)$
(1) set $\mathcal{C}_e = \emptyset$
(2) for every pair $(Q_{e_l}, Q_{e_r}) \in \mathcal{C}_{e_l} \times \mathcal{C}_{e_r}$, if $\mathbf{merge}(Q_{e_l}, Q_{e_r}) \neq \mathsf{void}$,
then let $\mathcal{C}_e \leftarrow \mathcal{C}_e \cup \{\mathbf{merge}(Q_{e_l}, Q_{e_r})\}$.
(3) return $\mathcal{C}_e$

It remains to describe the routine **merge**. For this, assume that it receives as inputs the (d,k,q)-configurations $Q_l = ((\mathcal{X}_l, \chi_l), (\mathcal{A}_l, \alpha_l), (\mathcal{F}_l, \mathcal{E}_l), \delta_l, z_l)$ and $Q_r = ((\mathcal{X}_r, \chi_r), (\mathcal{A}_r, \alpha_r), (\mathcal{F}_r, \mathcal{E}_r), \delta_r, z_r)$ of $(X_{e_l}, \mathbf{A}_{e_l})$ and $(X_{e_r}, \mathbf{A}_{e_r})$ respectively. Procedure $\mathbf{merge}(Q_{e_l}, Q_{e_r})$ returns a (d,k,q)-configuration $((\mathcal{X}, \chi), (\mathcal{A}, \alpha), (\mathcal{F}, \mathcal{E}), \delta, z)$ of $(X_e, \mathbf{A}_e)$ constructed as follows:

1. If $z_r + z_r > q$, then return void, otherwise $z = z_l + z_r$ (This controls the number of 1-edges that are now contained in Δ_e)
2. Let $(\mathcal{X}', \chi') = (\mathcal{X}_l, \chi_l) \oplus (\mathcal{X}_r, \chi_r)$ and if $\chi'^{-1}(2) \neq \emptyset$ then return void.
(This compute the "fusion" of the connected components of $(V(H_{e_l}, E^0_{e_l}))$ and $(V(H_{e_r}, E^0_{e_r}))$ with vertices in V_{e_l} and V_{e_r} and makes sure that none of the created components contains 2 or more 0-vertices.)
3. Let $(\mathcal{X}, \chi) = (\mathcal{X}'_l, \chi'_l)|_{V_e}$
(This computes the fusion $(\mathcal{X}'_l, \chi'_l)$ is restricted on the boundary O_e of Δ_e.)
4. Let $(\mathcal{A}', \alpha') = (\mathcal{A}_l, \alpha_l) \oplus (\mathcal{A}_r, \alpha_r)$ and if $\alpha'^{-1}(k+1) \neq \emptyset$ then return void.
5. Let $(\mathcal{A}, \alpha) = (\mathcal{A}_l, \alpha_l) \oplus (\mathcal{A}_r, \alpha_r)|_{\mathbf{A}_e}$.
6. Compute the function $\gamma : (\mathcal{F}_{e_l} \cup \mathcal{F}_{e_r} \cup X_e) \times (\mathcal{F}_{e_l} \cup \mathcal{F}_{e_r} \cup X_e) \to \{0, \ldots, d+1\}$, whose description is given latter.
7. Take the disjoint union of the graphs $(\mathcal{F}_l, \mathcal{E}_l)$ and $(\mathcal{F}_r, \mathcal{E}_r)$ and remove from it every edge $\{\phi_1, \phi_2\}$ for which $\gamma(\phi_1, \phi_2) \leq d$. Let $\mathcal{G}^+ = (\mathcal{F}^+, \mathcal{E}^+)$ be the obtained graph.

8. If for some edge $\{\phi_1, \phi_2\} \in \mathcal{E}^+$ it holds that for every $x_1, x_2 \in V_e$, $\gamma(\phi_1, x_1) + \gamma(\phi_2, x_2) > d$, then return void.
9. Consider the function $\lambda : \mathcal{F}_l \cup \mathcal{F}_r \to \{1, \ldots, d\}^{X_e}$ such that $\lambda(\phi) = \{(x, \gamma(\phi, x)) \mid x \in X_e\}$.
10. For every $\phi' \in \lambda(\mathcal{F}_l \cup \mathcal{F}_r)$, do the following for every set $\mathsf{F} = \lambda^{-1}(\phi')$: identify in $\mathcal{G}^+$ all vertices in F and if at least one pair of them is adjacent in $\mathcal{G}^+$, then add an loop on the vertex created after this identification. Let $\mathcal{G} = (\mathcal{F}, \mathcal{E})$ be the resulting graph (notice that $\mathcal{F} = \lambda(\mathcal{F}_l \cup \mathcal{F}_r)$).
11. $\delta = \{((x, x'), \gamma(x, x')) \mid x, x' \in V_e\}$.

The definition of function γ. We present here the definition of the function γ used in the above description of the tables of the dynamic programming procedure.

Given a non-empty set X and $q \in \{0, 1\}$ we define

$$\begin{aligned}\mathsf{ord}^q(X) \quad = \quad & \{\pi \mid \exists X' \subseteq X : X' \neq \emptyset \wedge \ |X'| \bmod 2 = q \\ & \wedge\ \pi \text{ is an ordering of } X'\}\end{aligned}$$

Given γ_l and γ_r, we define $\gamma : (\mathcal{F}_{e_l} \cup \mathcal{F}_{e_r} \cup X_e) \times (\mathcal{F}_{e_l} \cup \mathcal{F}_{e_r} \cup X_e) \to \{0, \ldots, d+1\}$ by distinguishing the following cases:

1. If $(x \in X_e \setminus X_{e_r} \wedge \phi \in \mathcal{F}_{e_l})$ or $(x \in X_e \setminus X_{e_l} \wedge \phi \in \mathcal{F}_{e_r})$, then
$$\begin{aligned}\gamma(\phi, x) \quad = \quad & \min\big\{\phi(x), \min\{\phi(p_1) + \sum_{[\![1,\rho-1]\!]} \delta_{\mathbf{s}(i)}(p_i, p_{i+1}) + \\ & \delta_{\mathbf{s}(\rho)}(p_\rho, x) \mid [p_1, \ldots, p_\rho] \in \mathsf{ord}^0(X_e^F)\}\big\},\end{aligned}$$
where $\mathbf{s}(i) =$ "l" if $\langle x \in X_e \setminus X_{e_l}\rangle = (i \bmod 2)$, otherwise $\mathbf{s}(i) =$ "r".
2. If $(x \in X_e \setminus X_{e_l} \wedge \phi \in \mathcal{F}_{e_l})$ or $(x \in X_e \setminus X_{e_r} \wedge \phi \in \mathcal{F}_{e_r})$, then
$$\begin{aligned}\gamma(\phi, x) \quad = \quad & \min\big\{\phi(p_1) + \sum_{[\![1,\rho-1]\!]} \delta_{\mathbf{t}(i)}(p_i, p_{i+1}) + \delta_{\mathbf{t}(\rho)}(p_\rho, x) \\ & \mid [p_1, \ldots, p_\rho] \in \mathsf{ord}^1(X_e^F)\}\big\},\end{aligned}$$
where $\mathbf{t}(i) =$ "l" if $\langle x \in X_e \setminus X_{e_l}\rangle \neq (i \bmod 2)$, otherwise $\mathbf{t}(i) =$ "r".
3. If x is one of the (at most two) vertices in $(X_{e_r} \cap X_{e_r}) \setminus X_e^F$ and $\phi \in \mathcal{F}_{e_l} \cup \mathcal{F}_{e_r}$, then
$$\begin{aligned}\gamma(\phi, x) \quad = \quad & \min\big\{\phi(x), \\ & \min\{\phi(p_1) + \sum_{[\![1,\rho-1]\!]} \delta_{\mathbf{u}(i)}(p_i, p_{i+1}) + \delta_{\mathbf{u}(q)}(p_\rho, x) \\ & \mid [p_1, \ldots, p_\rho] \in \mathsf{ord}^q(X_e^F)\} \mid q \in \{0, 1\}\big\}\end{aligned}$$
where $\mathbf{u}(i) =$ "r" if $\langle \phi \in \mathcal{F}_{e_l}\rangle = (i \bmod 2)$, otherwise $\mathbf{u}(i) =$ "l".
4. If $\phi, \phi' \in \mathcal{F}_l \cup \mathcal{F}_r$, then
$$\begin{aligned}\gamma(\phi, \phi') \quad = \quad & \min\big\{\phi(p_1) + \sum_{[\![1,\rho-1]\!]} \delta_{\mathbf{u}(i)}(p_i, p_{i+1}) + \phi'(p_\rho) \\ & \mid [p_1, \ldots, p_\rho] \in \mathsf{ord}^q(X_e^F)\big\}\end{aligned}$$
In this equality, $q = 1$ if ϕ and ϕ' belong in different sets in $\{\mathcal{F}_l, \mathcal{F}_r\}$, otherwise $q = 0$. The function $\mathbf{u}$ is the same as in the previous case.
5. If $x_1, x_2 \in X_e \setminus X_{e_r}$ or $x_1, x_2 \in X_e \setminus X_{e_l}$, then
$$\begin{aligned}\delta(x_1, x_2) \quad = \quad & \min\big\{\delta_{\mathbf{y}(0,x_1)}(x_1, x_2), \min\{\delta_{\mathbf{y}(0,x_1)}(x_1, p_1) + \\ & \sum_{i \in [\![1,\rho-1]\!]} \delta_{\mathbf{y}(i,x_1)}(p_i, p_{i+1}) + \\ & \delta_{\mathbf{y}(0,x_2)}(p_\rho, x_2) \mid [p_1, \ldots, p_\rho] \in \mathsf{ord}^0(X_e^F)\}\big\}\end{aligned}$$
In this equality $\mathbf{y}(i, x) =$ "l" if $\langle x \in X_e \setminus X_{e_r}\rangle = \langle i \bmod 2 = 0\rangle$ otherwise $\mathbf{y}(i, x) =$ "r".

6. If x_1, x_2 belong in different sets is $\{X_e \setminus X_{e_r}, X_e \setminus X_{e_l}\}$, then

$$\delta(x_1,x_2) = \min\Big\{\delta_{\mathbf{y}(0,x_1)}(x_1,p_1) + \sum_{[\![1,\rho-1]\!]} \delta_{\mathbf{y}(i,x_1)}(p_i,p_{i+1}) + \delta_{\mathbf{y}(0,x_2)}(p_\rho,x_2) \mid [p_1,\ldots,p_\rho] \in \mathsf{ord}^1(X_e^F)\Big\}$$

The function $\mathbf{y}$ is the same as in the previous case.

7. If exactly one, say x_2, of x_1, x_2 belongs in $X_{e_r} \cap X_{e_r}) \setminus X_e^F$, then

$$\delta(x_1,x_2) = \min\Big\{\delta_{\mathbf{y}(0,x_1)}(x_1,x_2), \min\Big\{\min\{\delta_{\mathbf{y}(0,x_1)}(x_1,p_1) + \sum_{[\![1,\rho-1]\!]} \delta_{\mathbf{y}(i,x_1)}(p_i,p_{i+1}) + \delta_{\mathbf{y}(0,x_2)}(p_\rho,x_2) \,|\, [p_1,\ldots,p_\rho] \in \mathsf{ord}^q(X_e^F)\} \,|\, q \in \{0,1\}\Big\}\Big\}$$

The function $\mathbf{y}$ is the same as in the two previous cases. In case x_1 belongs in $X_{e_r} \cap X_{e_r}) \setminus X_e^F$, then just swap the positions of x_1 and x_2 in the above equation.

8. If both x_1, x_2 belong in $X_{e_r} \cap X_{e_r}) \setminus X_e^F$, then

$$\delta(x_1,x_2) = \min\Big\{\delta_l(x_1,x_2), \delta_r(x_1,x_2), \min\{\min\{\delta_{\mathbf{z}(0,j)}(x_1,p_1) + \sum_{[\![1,\rho-1]\!]} \delta_{\mathbf{z}(i,j)}(p_i,p_{i+1}) + \delta_{\mathbf{z}(q,j)}(p_\rho,x_2) \,|\, [p_1,\ldots,p_\rho] \in \mathsf{ord}^q(X_e^F)\} \,|\, (q,j) \in \{0,1\}^2\}\Big\}$$

In the previous equality, $\mathbf{z}(i,j) =$ "l" if $(i+j \bmod 2) = 0$, otehrwise $\mathbf{z}(i,x) =$ "r".

Running time analysis. It now remains to prove that procedure **join** runs in $(\alpha(q))^2 \cdot 2^{O(k^2)+2^{O(b\log d)}}$ steps. Recall that there exists a function f such that $|\mathfrak{T}(e)| \leq f(k,q,b,d)$. Therefore **merge** will be called in Step (2) at most $(f(k,q,b,d))^2$ times. The first computationally non-trivial step of **merge** is Step 5, where function γ is computed. Notice that γ has at most $((d+1)^{|X_{e_l}|} + (d+1)^{|X_{e_r}|} + |X_e|)^2 = 2^{O(b\cdot\log d)}$ entries and each of their values require running over all permutations of the subsets of X_e^F that are at most $b! = 2^{O(b\cdot\log b)}$. These facts imply that the computation of γ takes $2^{O(b\cdot\log b)}$ steps. As Steps 6–10 deal with graphs of $2^{O(b\cdot\log d)}$ vertices, the running time of **join** is the claimed one. ◀

We are now in position to prove the main algorithmic result of this paper.

Proof of Theorem 2. Given an input $I = (G,q,k,d)$ of BBFPDC, we run the algorithm of Lemma 4 with G and k as input. Let $H = G_k$ be the output of this algorithm. From the same lemma, he construction of H takes $O(k^2 n)$ steps. Then we run the algorithm of Proposition 5 with (H,w) as input, where $w = c_1 \cdot c_2 \cdot k \cdot d$. If the answer is that $\mathbf{bw}(H) > w$, then, from Lemma 4, $\mathbf{tw}(G) > c_1 \cdot d$, therefore, from Lemma 3, we can safely report that I is a NO-instance. If the algorithm of Proposition 5 outputs a sphere-cut decomposition $D = (T,\mu)$ of width at most $w = O(k \cdot d)$ then we call the dynamic programming algorithm of Lemma 6, with input (G,H,q,k,d,D,b). This, from Lemma 4, provides an answer to BBFPDC for the instance I in $(\alpha(q))^2 \cdot 2^{O((kd)^2\log d)+2^{O((kd)\log d)}} \cdot n = (\alpha(q))^2 \cdot 2^{2^{O((kd)\log d)}} \cdot n$ steps and this completes the proof of the theorem. ◀

6 Discussion

We remark that our algorithm still works for the classic PDC problem when the face-degree of the input graph is bounded. For this we define the following problem:

Bounded Face BDC (FPDC)
Input: a plane graph G with face-degree at most $k \in \mathbb{N}_{\geq 3}$, and $d \in \mathbb{N}$
Question: is it possible to add edges in G such that the resulting embedding remains plane and has diameter at most d?

We directly have the following corollary of Theorem 2.

▶ **Theorem 7.** *It is possible to construct an* $O(n^3)+2^{2^{O((kd)\log d)}}\cdot n$*-step algorithm for* FPDC.

To construct an FPT-algorithm for PDC when parameterized by d remains an insisting open problem. The reason why our approach does not apply (at least directly) for PDC is that, as long as a completion may add an arbitrary number of edges in each face, we cannot guarantee that our dynamic programming algorithm will be applied on a graph of bounded branchwidth. We believe that our approach and, in particular, the machinery of our dynamic programming algorithm, might be useful for further investigations on this problem.

All the problems in this paper are defined on plane graphs. However, one may also consider the "non-embedded" counterparts of the problems PDC and BPDC by asking that their input is a planar combinatorial graphs (without a particular embedding). Similarly, such a counterpart can also be defined for the case of BFPDC if we ask whether the completion has an embedding with at most k new edges per face. Again, all these parameterized problems are known to be (non-constructively) in FPT, because of the results in [13, 11]. However, our approach fails to design the corresponding algorithms as it strongly requires an embedding of the input graph. For this reason we believe that even the non-embedded versions of BPDC and BFPDC are as challenging as the general Planar Diameter Completion problem.

Acknowledgement. We would like to thank the anonymous referees of an earlier version of this paper for their remarks and suggestions that improved the presentation of the paper.

References

1 F. R. K. Chung. Diameters of graphs: Old problems and new results. *Congressus Numerantium*, 60, 1987.
2 Bruno Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. *Handbook of Graph Grammars*, pages 313–400, 1997.
3 Italo J. Dejter and Michael R. Fellows. Improving the diameter of a planar graph. Manuscript, May 1993.
4 Frederic Dorn, Eelko Penninkx, Hans L. Bodlaender, and Fedor V. Fomin. Efficient exact algorithms on planar graphs: Exploiting sphere cut decompositions. *Algorithmica*, 58(3):790–810, 2010.
5 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity.* Texts in Computer Science. Springer, 2013.
6 Yong Gao, Donovan R. Hare, and James Nastos. The parametric complexity of graph diameter augmentation. *Disc. Appl. Math.*, 161(10-11):1626–1631, 2013.
7 Petr A. Golovach, Clément Requilé, and Dimitrios M. Thilikos. Variants of plane diameter completion. *CoRR*, abs/1509.00757, 2015.
8 Qian-Ping Gu and Hisao Tamaki. Optimal branch-decomposition of planar graphs in $O(n^3)$ time. *ACM Transactions on Algorithms*, 4(3), 2008.
9 Chung-Lun Li, S. Thomas McCormick, and David Simchi-Levi. On the minimum-cardinality-bounded-diameter and the bounded-cardinality-minimum-diameter edge addition problems. *Oper. Res. Lett.*, 11(5):303–308, 1992.

10 David Lichtenstein. Planar formulae and their uses. *SIAM Journal on Computing*, 11(2):329–343, 1982.

11 Neil Robertson and Paul D. Seymour. Graph minors XIII. The disjoint paths problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995.

12 Neil Robertson and Paul D. Seymour. Graph Minors. XIII. The disjoint paths problem. *J. Combin. Theory, Ser. B*, 63(1):65–110, 1995.

13 Neil Robertson and Paul D. Seymour. Graph minors. XX. Wagner's conjecture. *J. Combin. Theory Ser. B*, 92(2):325–357, 2004.

14 Anneke A. Schoone, Hans L. Bodlaender, and Jan van Leeuwen. Diameter increase caused by edge deletion. *Journal of Graph Theory*, 11(3):409–427, 1987.

15 Paul D. Seymour and Robin Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994.

Parameterized and Approximation Algorithms for the Load Coloring Problem

Florian Barbero, Gregory Gutin, Mark Jones, and Bin Sheng

Royal Holloway, University of London
Egham, Surrey TW20 0EX, UK

Abstract

Let c, k be two positive integers. Given a graph $G = (V, E)$, the c-Load Coloring problem asks whether there is a c-coloring $\varphi : V \to [c]$ such that for every $i \in [c]$, there are at least k edges with both endvertices colored i. Gutin and Jones (IPL 2014) studied this problem with $c = 2$. They showed 2-Load Coloring to be fixed-parameter tractable (FPT) with parameter k by obtaining a kernel with at most $7k$ vertices. In this paper, we extend the study to any fixed c by giving both a linear-vertex and a linear-edge kernel. In the particular case of $c = 2$, we obtain a kernel with less than $4k$ vertices and less than $8k$ edges. These results imply that for any fixed $c \geq 2$, c-Load Coloring is FPT and the optimization version of c-Load Coloring (where k is to be maximized) has an approximation algorithm with a constant ratio.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases load Coloring, fixed-parameter tractability, kernelization

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.43

1 Introduction

Given a graph $G = (V, E)$ and an integer k, the 2-Load Coloring Problem introduced in [1], asks whether there is a coloring $\varphi : V \to \{1, 2\}$ such that for $i = 1$ and 2, there are at least k edges with both endvertices colored i. This problem is NP-complete [1], and Gutin and Jones studied its parameterization by k [9]. They proved that 2-Load Coloring is fixed-parameter tractable (FPT)[1] by obtaining a kernel with at most $7k$ vertices. It is natural to extend 2-Load Coloring to any number c of colors as follows. Henceforth, for a positive integer p, $[p] = \{1, 2, \ldots, p\}$.

▶ **Definition 1** (c-Load Coloring)**.** Let c be a positive integer. Given a positive integer k and a graph $G = (V, E)$, the c-Load Coloring problem asks whether there is a c-coloring $\varphi : V \to [c]$ such that for every $i \in [c]$, there are at least k edges with both endvertices colored i. If such a coloring φ exists, we call φ a (c, k)-coloring of G and we write $G \in (c, k)$-LC.

The c-Load Coloring problem can be viewed as a subgraph packing problem: decide whether a graph G contains c disjoint k-edge subgraphs.

Observe first that $G \in (1, k)$-LC if and only if $|E(G)| \geq k$. In this paper, we consider c-Load Coloring parameterized by k for every fixed $c \geq 2$. Note that c-Load Coloring is NP-complete for every fixed $c \geq 2$. Indeed, we can reduce 2-Load Coloring to c-Load Coloring with $c > 2$ by taking the disjoint union of G with $c - 2$ stars $K_{1,k}$.

[1] For comprehensive introductions to parameterized algorithms and complexity, see recent monographs [4, 7]; [10, 11] are excellent recent survey papers on kernelization.

10th International Symposium on Parameterized and Exact Computation (IPEC 2015).
Editors: Thore Husfeldt and Iyad Kanj; pp. 43–54

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

We prove that the problem admits a kernel with less than $2ck$ vertices. Thus, for $c = 2$ we improve the kernel result of [9]. To show our result, we introduce reduction rules, which are new even for $c = 2$. We prove that the reduction rules can run in polynomial time and that an irreducible graph with at least $2ck$ vertices is in (c, k)-LC.

While there are many parameterized graph problems which admit kernels linear in the number of vertices, usually only problems on classes of sparse graphs admit kernels linear in the number of edges (since in such graphs the number of edges is linear in the number of vertices), see, e.g., [3, 7, 11]. To the best of our knowledge, only trivial $O(k)$-edge kernels for general graphs have been described in the literature, e.g., the kernel for Max Cut parameterized by solution size (Prieto [12] improved the trivial result by obtaining a kernel with at most $2k$ edges and at most k vertices). Thus, our next result is somewhat unusual: c-Load Coloring admits a kernel with $O(k)$ edges for every fixed $c \geq 2$. Namely, the kernel has less than $8k$ edges when $c = 2$ and less than $6.25c^2k$ edges when $c > 2$.

The optimization version of c-Load Coloring is as follows: for a graph G, find the maximum k such that $G \in (c, k)$-LC. We show that because of the above bounds on the number of edges in a kernel, this optimization problem, called the Max c-Load Coloring problem, admits constant ratio approximation algorithms for any fixed c.

The paper is organized as follows. After providing additional terminology and notation on graphs in the remainder of this section, we show that the problem admits a kernel with less than $2ck$ vertices in Section 2. Then, in Section 3, we prove an upper bound on the number of edges in a kernel for every $c \geq 2$ and the corresponding approximation result for Max c-Load Coloring. We improve our bound for $c = 2$ in Section 4. The bound implies the approximation ratio of $4 + \varepsilon$ for every $\varepsilon > 0$. We complete the paper with discussions in Section 5.

Graphs. For a graph G, $V(G)$ ($E(G)$, respectively) denotes the vertex set (edge set, respectively) of G, $\Delta(G)$ denotes the maximum degree of G, n its number of vertices, and m its number of edges. A vertex u with degree 0 (1, respectively) is an *isolated vertex* (a *leaf-neighbor of* v, where $uv \in E(G)$, respectively). For a vertex x and a vertex set X in G, $N(x) = \{y : xy \in E(G)\}$ and $N_X(x) = N(x) \cap X$. For disjoint vertex sets X, Y of G, let $G[X]$ be the subgraph of G induced by X, $E(X) = E(G[X])$ and $E(X, Y) = \{xy \in E(G) : x \in X, y \in Y\}$. For a coloring φ, we say that an edge uv is *colored* i if $\varphi(u) = \varphi(v) = i$.

2 Bounding Number of Vertices in Kernel

In this section, we show that c-Load Coloring admits a kernel with less than $2ck$ vertices. The fact that $(ck - 1)K_2$ is a No-instance suggests that this bound is likely to be optimal.

For any integer $i \geq 1$ and $\tau \in \{<, \leq, =, >, \geq\}$, $K_{1,\tau i}$ denotes a *star* $K_{1,j}$ such that $j \ \tau \ i$ and $j \geq 1$. For instance, $K_{1,\leq p}$ is a star with q edges, $q \in [p]$. Then, a *$K_{1,\tau i}$-graph* is a forest in which every component is a star $K_{1,\tau i}$, and a *$K_{1,\tau i}$-cover* of G is a spanning subgraph of G which is a $K_{1,\tau i}$-graph. We call any $K_{1,\tau i}$-graph a *star graph* and any $K_{1,\tau i}$-cover a *star cover*.

We first prove the bound for star graphs with small maximum degree.

▶ **Lemma 2.** *If G is a $K_{1,<2k}$-graph with $n \geq 2ck$, then $G \in (c, k)$-LC.*

Proof. Let G be a $K_{1,<2k}$-graph with $n \geq 2ck$. We prove the lemma by induction on c. The base case of $c = 1$ holds since a $K_{1,<2k}$-graph has no isolated vertices. Indeed, this property implies G has at least $\frac{V(G)}{2} \geq k$ edges.

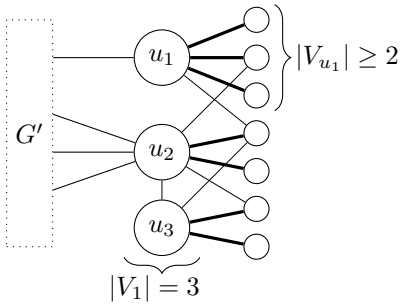


Figure 1 An overload from $O_{3,2}$.

Since all components of G are trees, for each one the number of vertices is one more than the number of edges. If there is a component C, with $k \leq |E(C)| < 2k$, we may color $V(C)$ with the same color (then, $G[V(C)] \in (1,k)$-LC). Since we only used $|V(C)| \leq 2k$ vertices, $H = G - V(C)$ has at least $2(c-1)k$ vertices and so $H \in (c-1,k)$-LC by the induction hypothesis. Thus, $G \in (c,k)$-LC.

We may assume that every component has less than k edges and let $C_1, \ldots, C_t$ be the components of G. Let b be the minimum nonnegative integer for which there exists $I \subseteq [t]$ such that $\Sigma_{i \in I}|E(C_i)| = k + b \geq k$. Since there is no isolated vertex in a star graph, $m \geq n/2 \geq ck$, and thus such a set I exists. Observe that for any $i \in I$, $|E(C_i)| > b$, as otherwise $\Sigma_{j \in I \setminus \{i\}}|E(C_j)| = k+b-|E(C_i)| \geq k$, a contradiction to the minimality of b. Since every component has less than k edges, $b \leq k-2$. For a star (V,E), the ratio $\frac{|V|}{|E|}$ increases when $|E|$ decreases. Thus, we have $\Sigma_{j \in I}|V(C_j)| \leq \Sigma_{j \in I}|E(C_j)| \max_{h \in I}(\frac{|V(C_h)|}{|E(C_h)|}) \leq (k+b)\frac{b+2}{b+1}$. But $2k-(k+b)\frac{b+2}{b+1} = \frac{(k-2-b)b}{b+1} \geq 0$, and so $\Sigma_{j \in I}|V(C_j)| \leq 2k$. We may color the components C_i, $i \in I$, by the same color. Again, we have that $H = G - V(\bigcup_{i \in I} C_i)$ has at least $2(c-1)k$ vertices and so $H \in (c-1,k)$-LC by the induction hypothesis. Thus, $G \in (c,k)$-LC. ◀

Since $G \in (c,k)$-LC whenever G has a subgraph $H \in (c,k)$-LC, we have that any graph with $n \geq 2ck$ and a $K_{1,<2k}$-cover is in (c,k)-LC. To decide the second property, we introduce a family $(O_{i,k})_{i,k \in \mathbb{N}}$ of overloads.

▶ **Definition 3.** We call a pair (V_1, V_2) of disjoint vertex sets an *overload from* $O_{i,k}$ if $|V_1| = i$, $N(v) \subseteq V_1$ for all $v \in V_2$, and for every $u \in V_1$ there is a set $V_u \subseteq N_{V_2}(u)$ such that $|V_u| \geq k$ and for every pair u, v of distinct vertices of V_1, $V_u \cap V_v = \emptyset$ (see Fig. 1).

Note that if v is an isolated vertex, the pair $(\emptyset, \{v\})$ is an overload from $O_{0,k}$. If a graph G has an overload (V_1, V_2) from $O_{i,k}$, then $G[V_1 \cup V_2] \in (i,k)$-LC: for each $u \in V_1$, color $V_u \cup \{u\}$ with one color. However, $G[V_1 \cup V_2] \notin (i+1,k)$-LC. Indeed, an edge can only be colored with one of $|V_1| = i$ colors. So, in any coloring, an overload from $O_{i,k}$ may give k edges for each of i colors but cannot bring any edge for all the other colors. From this observation, we deduce the following set of reduction rules. (Note that our reduction rules generalize the well-known Crown Reduction Rule. Similar, but different, reduction rules were used in [8].)

Reduction rule $\mathbf{R_{i,k}}$. If an instance G for (c,k)-LC contains an overload (V_1, V_2) from $O_{i,k}$, delete all the vertices of $V_1 \cup V_2$ from G and decrease c by i.

Since the existence of an overload from $O_{i,k}$ for $i \geq c$, in a graph G implies $G \in (c,k)$-LC, we only consider $R_{i,k}$ for $i < c$. We now show rules $R_{i,k}$ are safe and can be applied in time polynomial in n (recall that c is fixed). We say that a graph is *irreducible for* (c,k)*-LC* if it is not possible to apply any rule $R_{i,k}$, $i < c$, to the graph.

▶ **Lemma 4.** *Let G be a graph and G' be the graph obtained from G after applying reduction rule $R_{i,k}$. Then $G \in (c,k)$-LC if and only if $G' \in (c-i,k)$-LC.*

Proof. Let (V_1, V_2) be the overload from $O_{i,k}$ used to map (G, c) to $(G', c-i)$. On the one hand, if $G' \in (c-i,k)$-LC, there exists a $(c-i,k)$-coloring of G' and together with a (i,k)-coloring of the overload, we obtain a (c,k)-coloring of G: $G \in (c,k)$-LC. On the other hand, observe that the vertices of V_2 are isolated in $G - V_1$. Thus $E(G - V_1) = E(G - V_1 - V_2) = E(G')$. If $G \in (c,k)$-LC, in any (c,k)-coloring of G, there are at least $c - |V_1| = c - i$ colors with no edge with endvertices in V_1. These colors must have their k edges in $E(G - V_1) = E(G')$. Thus $G' \in (c-i,k)$-LC. ◀

▶ **Lemma 5.** *One can decide whether Rule $R_{i,k}$ is applicable to G in time $O(n^{i+O(1)})$.*

Proof. Generate all i-size subsets V_1 of $V(G)$. For each V_1, construct the set V_2 that includes every vertex outside V_1 whose only neighbors are in V_1. If $|V_2| \geq ik$, construct the following bipartite graph B: the partite sets of B are V_1' and V_2, where V_1' contains k copies of every vertex v of V_1 with the same neighbors as v. Observe that B has a matching covering V_1' if and only if $R_{i,k}$ can be applied to G for the overload (V_1, V_2). It is not hard to turn the above into an algorithm of runtime $O(n^{i+O(1)})$. ◀

In fact, the running time in Lemma 5 can be improved: in the journal version of this paper, we will show that $O(n^{i+O(1)})$ can be replaced by $O((cn)^2)$.

Now, we want to show that a graph without any overloads has a star cover with small degree. If so, since an irreducible graph has no overload from $O_{i,k}, i \in [c-1]$, an irreducible graph for (c,k)-LC with at least $2ck$ vertices would be in (c,k)-LC:

▶ **Lemma 6.** *Let G be a graph and k a positive integer. If G has no overload from $O_{i,k}$ for any $i \leq n$, then G has a $K_{1,\leq\max\{3,k\}}$-cover.*

Proof. Let G be a graph with no overload from $O_{i,k}$ for any $i \leq n$. We first show that G has a star cover. Since it is not possible to apply $R_{0,k}$, G has no isolated vertex. By choosing a spanning tree of each component of G, we obtain a forest F. If a tree in F is not a star, it has an edge between two non-leaves. As long as F contains such an edge, delete it from F. Observe that F becomes a star cover of G. However, the number of leaves in each star of F is only bounded by $\Delta(G)$. We will show that among the possible star covers of G, there exists a $K_{1,\leq\max(3,k)}$-cover.

For each star cover F, we define the *F-sequence* $(F_{\Delta(G)}, F_{\Delta(G)-1}, \ldots, F_1)$, where F_i is the number of stars with exactly i edges, $i \in [\Delta(G)]$. We say a star cover F is *smaller* than a star cover F' if and only if the *F-sequence* is smaller than the *F'-sequence* lexicographically, i.e. there exists some $i \in [\Delta(G)]$ such that $F_i < F_i'$ and for every $j > i$, $F_j = F_j'$.

We select a star cover S of G which has the lexicographically minimum sequence, that is, for any star cover F of G, the S-sequence is smaller or equal to the F-sequence. Suppose that $\Delta(S) > \max\{3,k\}$. Let C_i (L_i, respectively) be the set of all the centers (leaves, respectively) of all stars of S isomorphic to $K_{1,i}$. We also define $L_{\geq i} = \cup_{j \geq i} L_j$. We will now prove two claims.

Claim 1. *There is no edge $uv \in E(G) \setminus E(S)$ such that $u \in L_{\geq 3}$ and $v \in L_{\geq 1}$.*

Indeed, suppose there exists one and let x, y be such that $xu, yv \in E(S)$. If $v \in L_{\geq 2}$, then by deleting edges xu, yv and adding edge uv, we do not create any isolated vertex but we decrease the size of the stars centered at x and y, and thus we get a smaller star cover than S, a contradiction. Otherwise, v is an endvertex of an independent edge, and by deleting

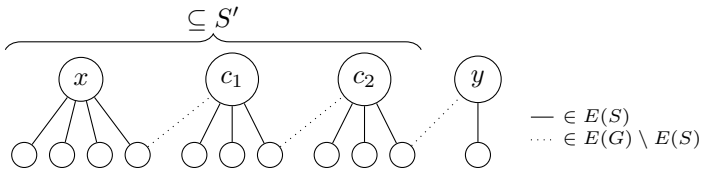


Figure 2 An alternating path from x to y with $\Delta(S) = 4$.

edge xu and adding edge uv, we decrease the size of the star centered at x, and create a star $K_{1,2}$ centered at v, which still induces a star cover smaller than S, a contradiction.

Claim 2. *Suppose S contains a star isomorphic to $K_{1,i}$ and centered at vertex x, and a star isomorphic to $K_{1,j}$ and centered at vertex y, such that $i - j \geq 2$. There is no path from x to y in which the odd edges are in $E(S)$ and go from a center to a leaf, and the even edges are in $E(G) \setminus E(S)$ and go from a leaf to a center. (see Fig. 2)*

Suppose there exists such a path. Then by deleting the odd edges of the path and adding the even ones, we do not create isolated vertices because x still has leaf-neighbors, y gets a neighbor, every transitional center keeps the same number of leaf-neighbors and the transitional leaves always go to a new center. This operation only decreases the size of star centered at x by 1 and increases the size of star centered at y by 1, giving us a lexicographically smaller star cover, a contradiction.

Let S' be the subgraph of S containing all stars $K_{1,\Delta(S)}$ of S. While there is an edge $uv \in E(G) \setminus E(S)$ such that u is a leaf of S' and $v \in C_{\Delta(S)-1} \setminus S'$, we add the star centered at v to S'. Let C' (L', respectively) be the centers (leaves, respectively) in S'. Suppose there is an edge $uv \in E(G) \setminus E(S)$ such that $u \in L' \subseteq L_{\geq \Delta(S)-1} \subseteq L_{\geq 3}$ and $v \in V(G) \setminus C'$. By Claim 1, $v \notin L_{\geq 1}$. Since $v \notin C_{\Delta(S)} \subseteq C'$ and since the above procedure has terminated, $v \in C_j$ for some j such that $\Delta(S) - j \geq 2$. Now, by construction, there is an alternating path from a vertex in $C_{\Delta(S)}$ to a vertex in C_j of the type described in Claim 2, which is impossible.

So, there is no edge $uv \in E(G) \setminus E(S)$ such that $u \in L'$ and $v \notin C'$. This means that for any $u \in L', N(u) \subseteq C'$. Furthermore, for each $u \in C'$, we can define V_u to be the leaves of the star centered at u, for which we have $|V_u| \geq \Delta(S) - 1 \geq k$. Thus, (C', L') is an overload from $O_{|C'|,k}$, which is impossible. ◀

Since we obtain the expected result, we can deduce our theorem:

▶ **Theorem 7.** *For $k > 1$, if G is irreducible for (c,k)-LC and has at least $2ck$ vertices, then $G \in (c,k)$-LC. Furthermore, for any fixed $c \geq 2$ and for any positive integer k, c-Load Coloring admits a kernel with less than $2ck$ vertices.*

Proof. Observe first that for every $c \geq 2$, $G \in (c,1)$-LC if and only if G has a matching with at least c edges. Since this property can be decided in polynomial time, we just need to consider the case when $k > 1$.

By Lemmas 4 and 5, there is a polynomial algorithm that reduces an instance (G,c) to an instance (G',c') such that $c' \leq c$ and G' is irreducible for (c',k)-LC. Suppose the kernel G' has at least $2c'k$ vertices, but $G \notin (c',k)$-LC. Then, observe that G' does not have an overload from $O_{i,k}$, $i \geq c'$, and thus G' has a $K_{1,\leq \max(3,k)}$-cover by Lemma 6. Since $k > 1$, this star cover is a $K_{1,<2k}$-cover and Lemma 2 implies that $G' \in (c',k)$-LC, a contradiction. So, if $|V(G')| \geq 2c'k$, then $G' \in (c',k)$-LC and $G \in (c,k)$-LC, hence we may conclude the kernel G' has less than $2c'k \leq 2ck$ vertices. ◀

3 Bounding Number of Edges in Kernel

Let $S(c)$ be the integer sequence defined by induction by $S(1) = 1$, $S(2c) = 4S(c)$ and $S(2c+1) = 2S(c)+2S(c+1)$. This sequence is known as A073121 in the Online Encyclopedia of Integer Sequences [13] (see also [2]). We will use the following technical result.

▶ **Lemma 8.** *If c is even, $S(c) \leq \frac{9c^2-4}{8}$. For arbitrary c, $S(c) \leq \frac{9c^2-1}{8}$.*

Proof. It is easy to check the base cases: $S(1) = 1 = \frac{9(1)^2-1}{8}$, $S(2) = 4 = \frac{9(2)^2-4}{8}$ and $S(3) = 10 = \frac{9(3)^2-1}{8}$. We now assume the claim holds for every $c \leq 2c' - 1$ and we will prove it for $c = 2c'$ and $c = 2c' + 1$.

For even value, we have:

$$S(2c) = 4S(c) \leq 4\frac{9c^2-1}{8} = \frac{9(2c)^2-4}{8}.$$

For odd value, we have:

$$\begin{aligned} S(2c+1) &= 2(S(c) + S(c+1)) \\ &\leq 2\frac{9c^2 + 9(c+1)^2 - 1 - 4}{8} = \frac{9(2c+1)^2-1}{8}. \end{aligned}$$

◀

By using the kernel we proved in the previous section, we show that c-Load Coloring admits a kernel with less than $(2S(c) + 4c^2 - 5c)k$ edges. Because of the upper bound on $S(c)$ given by Lemma 8, the number of edges in a kernel may be bounded by $6.25c^2k$. We first prove a smaller bound for bipartite graphs.

▶ **Lemma 9.** *Let $b(c, k, n) = S(c)k + (c - 1)n$. For every positive integer c and bipartite graph G with n vertices, if $m \geq b(c, k, n)$ then $G \in (c, k)$-LC.*

Proof. We prove the lemma by induction on c. For the base case, observe that any graph with at least $k = b(1, k, n)$ edges is in $(1, k)$-LC for every k and n. We now assume the claim holds for every $c \leq 2c' - 1$ and we will prove it for $c = 2c'$ and $c = 2c' + 1$.

Suppose that $G = (A \cup B, E)$ is a bipartite graph with n vertices and at least $b(c, k, n)$ edges, but $G \notin (c, k)$-LC. Let B_2 be a maximal subset of B such that

$$|E(A, B_2)| < b(c - c', k, |A| + |B_2|) + b(c - c', k, |B_2|)\,. \tag{1}$$

So, for any vertex $u \in B \setminus B_2$, the set $B_2 \cup \{u\}$ doesn't satisfy (1). Such a set B_2 exists since the empty set satisfies (1). Moreover, for any partition (A_1, A_2) of A, we know there exists $i \in \{1, 2\}$ such that

$$|E(A_i, B_2 \cup \{u\})| \geq b(c - c', k, |A_i| + |B_2 \cup \{u\}|) \tag{2}$$

as otherwise, the linearity in n of $b(c, k, n)$ implies a contradiction with the maximality of B_2:

$$\begin{aligned} |E(A, B_2 \cup \{u\})| &= |E(A_1, B_2 \cup \{u\})| + |E(A_2, B_2 \cup \{u\})| \\ &< b(c - c', k, |A_1| + |B_2 \cup \{u\}|) + b(c - c', k, |A_2| + |B_2 \cup \{u\}|) \\ &= b(c - c', k, |A| + |B_2 \cup \{u\}|) + b(c - c', k, |B_2 \cup \{u\}|). \end{aligned}$$

Let $B_1 = B \setminus B_2$, $A_1 = A$ and $A_2 = \emptyset$. We define the following inequalities.

$$|E(A_1, B_1)| < b(c', k, |A_1| + |B_1|) + |A_1| \tag{3}$$

$$|E(A_2, B_1)| < b(c', k, |A_2| + |B_1|) + |A_2|. \tag{4}$$

While (3) does not hold and (4) holds, we move an arbitrary vertex from A_1 to A_2. Suppose eventually (3) and (4) are both false and let u be an arbitrary vertex in B_1. We deduce for both $i = 1$ and $i = 2$ that

$$|E(A_i, B_1 \setminus \{u\})| \geq b(c', k, |A_i| + |B_1|)\,.$$

Thus, there exist disjoint vertex sets X and Y such that $|E(X)| \geq b(c', k, |X|)$ and $|E(Y)| \geq b(c-c', k, |Y|)$ (either $X = A_1 \cup B_1 \setminus \{u\}$ and $Y = A_2 \cup B_2 \cup \{u\}$, or $X = A_2 \cup B_1 \setminus \{u\}$ and $Y = A_1 \cup B_2 \cup \{u\}$), depending on whether (2) holds for $i = 1$ or $i = 2$. By taking a (c', k)-coloring of X and a $(c-c', k)$-coloring of Y, we have that $G \in (c, k)$-LC, a contradiction.

So, we may assume (3) eventually holds. If $A_2 = \emptyset$, then $|E(A_2, B_1)| = 0$. Otherwise, let v be the last vertex moved from A_1 to A_2. Observe that

$$\begin{aligned} |E(A_2, B_1)| &\leq |E(A_2 \setminus \{v\}, B_1)| + |B_1| \\ &< b(c', k, |A_2 \setminus \{v\}| + |B_1|) + |A_2 \setminus \{v\}| + |B_1| \text{ (by (4))}. \\ &< b(c', k, |A_2| + |B_1|) + |A_2| + |B_1|. \end{aligned} \tag{5}$$

In both cases, (5) holds and we can bound the number of edges in G:

$$\begin{aligned} |E(G)| &= |E(A, B_2)| + |E(A_1, B_1)| + |E(A_2, B_1)| \\ &< b(c - c', k, |A| + |B_2|) + b(c - c', k, |B_2|) \\ &+ b(c', k, |A_1| + |B_1|) + |A_1| \\ &+ b(c', k, |A_2| + |B_1|) + |A_2| + |B_1| \\ &\quad \text{(by inequalities (1),(3),(5))}. \end{aligned}$$

If $c = 2c'$, we have $c - c' = c'$ and it is not hard to check that

$$|E(G)| < 4S(c')k + 2(c' - 1)n + n = b(c, k, n).$$

Otherwise, $c = 2c' + 1$ and then $c - c' = c' + 1$. Thus,

$$\begin{aligned} |E(G)| &< 2S(c')k + 2S(c' + 1)k + 2(c' - 1)n \\ &\quad +|A| + 2|B_2| + |A_1| + |A_2| + |B_1| \\ &\leq S(2c' + 1)k + 2c'n = b(c, k, n). \end{aligned}$$

Thus, for $c = 2c'$ and $c = 2c' + 1$, we have $|E(G)| < b(c, k, n)$, a contradiction. So, there is no bipartite graph with n vertices and at least $b(c, k, n)$ edges such that $G \notin (c, k)$-LC. ◀

We now generalize this lemma for any graph. We would like to find a partition (A, B) of V such that $|E(A)| + |E(B)|$ is bounded, since $|E(A, B)|$ is bounded.

▶ **Lemma 10.** *Let $f(c, k, n) = (2S(c) - c)k + 2(c - 1)n$. For every positive integer c and every graph G with n vertices, if $m \geq f(c, k, n)$ then $G \in (c, k)$-LC.*

Proof. We prove the lemma by induction on c. For the base case, observe that any graph with at least $k = f(1, k, n)$ edges is in $(1, k)$-LC for every k and n. We now assume the claim holds for every $c \leq 2c' - 1$ and we will prove it for $c = 2c'$ and $c = 2c' + 1$.

Consider a graph G with n vertices and at least $f(c', k, n)$ edges, such that $G \notin (c, k)$-LC. We will first show that there exists a set $A \subseteq V(G)$ such that $f(c', k, |A|) \leq |E(A)| \leq f(c', k, |A|) + |A|$ (and thus $G[A] \in (c', k)$-LC). We may construct the set A as follows:

initially $A = \emptyset$ and while $|E(A)| < f(c', k, |A|)$, add an arbitrary vertex of $V(G) \setminus A$ to A. Let u be the last added vertex. Then

$$|E(A)| \leq |E(A \setminus \{u\})| + |A \setminus \{u\}| < f(c', k, |A \setminus \{u\}|) + |A \setminus \{u\}| < f(c', k, |A|) + |A|.$$

Let $B = V(G) \setminus A$. If $G[B] \in (c - c', k)$-LC, then $G \in (c, k)$-LC, a contradiction. So $|E(B)| < f(c - c', k, |B|)$. Furthermore, $|E(A, B)| < b(c, k, n)$ by Lemma 9. Finally, we may bound $|E(G)|$. If $c = 2c'$, we have $c - c' = c'$

$$\begin{aligned} |E(G)| &< f(c', k, |A|) + f(c', k, |B|) + b(2c', k, n) + |A| \\ &\leq (2S(2c') - 2c')k + (4c' - 2)n = f(c, k, n). \end{aligned}$$

Otherwise, $c = 2c' + 1$ and $c - c' = c' + 1$. Thus,

$$\begin{aligned} |E(G)| &< f(c', k, |A|) + f(c' + 1, k, |B|) + b(2c' + 1, k, n) + |A| \\ &\leq (2S(2c' + 1) - (2c' + 1))k + 4c'n = f(c, k, n). \end{aligned}$$

Thus, in both cases $|E(G)| < f(c, k, n)$, as required. ◀

▶ **Theorem 11.** *The c-Load Coloring Problem admits a kernel with less than $f(c, k, 2ck) < 6.25c^2k$ edges.*

Proof. By Theorem 7, we can get a kernel with less than $2ck$ vertices. Thus by Lemmas 10 and 8, we get a kernel such that $|E(G)| < f(c, k, 2ck) < 6.25c^2k$. ◀

The size of this kernel may be optimal up to a constant factor. Indeed, the complete bipartite graph $K_{c,ck-1}$ is an irreducible graph for (c, k)-LC with $c^2k - c = O(c^2k)$ edges, but $K_{c,ck-1} \notin (c, k)$-LC. We can increase this lower bound by joining all c vertices on the smaller side of $K_{c,ck-1}$. The resulting graph is not in (c, k)-LC either, and it has $c^2k + \frac{c(c-3)}{2}$ edges.

We now consider an approximation algorithm for the Max c-Load Coloring problem: Given a graph G and integer c, we wish to determine the maximum k, denoted k_{opt}, for which $G \in (c, k)$-LC. Given an approximation algorithm, we define the approximation ratio $r(c) = \frac{k_{opt}}{k}$, where k is the output of the approximation algorithm.

Let $K(c)k$ be an upper bound of the number of edges in a kernel for (c, k)-LC and let $P(c) = \prod_{i=1}^{c} \frac{K(i)}{i}$. By Theorem 11, we may have $K(c) = 6.25c^2$.

▶ **Theorem 12.** *There is a $2^{c-1}P(c)$-approximation algorithm for* Max c-Load Coloring.

Proof. We prove the claim by induction on c. For $c = 1$, we have $P(1) = 1$. Assume the theorem is true for all $c' < c$ and let G be an instance for c-Load Coloring with n vertices and m edges. We may assume that G has no isolated vertices. Clearly, $k_{opt} \leq \frac{m}{c}$. Consider $k = \lfloor \frac{m}{K(c)} \rfloor$.

If $k = 0$, then $m < K(c)$ and we can find k_{opt} in $O(1)$ time.

Now let $k > 0$. If $n \leq 2ck$, then by the proof of Theorem 11, since $m \geq K(c)k$, $G \in (c, k)$-LC. So we return k, and $\frac{k_{opt}}{k} \leq \frac{m}{ck} \leq \frac{K(c)(k+1)}{ck} \leq \frac{2K(c)}{c} \leq 2^{c-1}P(c)$.

If $n \geq 2ck$ and G is irreducible for (c, k)-LC, then by Theorem 7, $G \in (c, k)$-LC and we return k as above. If $n \geq 2ck$ and G is not irreducible for (c, k)-LC, we can use Lemma 5 to reduce (G, c) to (G', c') with $c' < c$. By induction we may find k' such that $k'_{opt} \leq 2^{c'-1}P(c')k'$, where k'_{opt} is the optimal solution for Max c'-Load Coloring on G'. Now consider three cases:

- $k' \geq k$. Then $G' \in (c', k)$-LC and so $G \in (c, k)$-LC. This case also leads to the above conclusion.

- $k'_{opt} \leq 2^{c'-1}P(c')k' < k$. Because $k'_{opt} + 1 \leq k$, an overload from $O_{c-c',k}$ is also an overload from $O_{c-c',k'_{opt}+1}$, therefore G' can be derived from G using a reduction rule for $(c, k'_{opt} + 1)$-LC. Since $G' \notin (c', k'_{opt} + 1)$-LC, $G \notin (c, k'_{opt} + 1)$-LC. Thus $k_{opt} = k'_{opt}$. The algorithm may output k' which satisfies $k_{opt} = k'_{opt} \leq 2^{c'-1}P(c')k' \leq 2^{c-1}P(c)k$.
- $k' < k \leq 2^{c'-1}P(c')k'$. The algorithm gives k' as an approximation of k_{opt}. Then $\frac{k_{opt}}{k'} \leq \frac{m}{ck'} \leq \frac{K(c)(k+1)}{ck'} \leq \frac{K(c)}{c}\frac{2k}{k'} \leq \frac{K(c)}{c}2^{c'}P(c') \leq 2^{c-1}P(c)$.

In every case, the approximation ratio is at most $2^{c-1}P(c)$. ◀

4 Number of Edges in Kernel for $c = 2$

In this section, we look into the edge kernel problem for the special case when $c = 2$. By doing a refined analysis, we will give a kernel with less than $8k$ edges for $(2, k)$-LC, which is a better bound than the general one. Henceforth, we assume that G is irreducible for $(2, k)$-LC, and just consider the case when $|V(G)| < 4k$, as we have proved that if $|V(G)| \geq 4k$ then $G \in (2, k)$-LC.

▶ **Lemma 13.** *If G has at least $3k - 2$ edges and every component in G has less than k edges then $G \in (2, k)$-LC.*

Proof. We consider colorings of the graph such that vertices in the same component are colored with the same color. Thus every edge in the graph is colored with 1 or 2. Denote the set of edges colored i with $E_i, i = 1, 2$. Among all possible colorings, choose a coloring of the graph such that $|E_1| \geq |E_2|$ and $||E_1| - |E_2||$ is minimum. Suppose $|E_2| \leq k - 1$, then $|E_1| \geq 2k - 1$, $||E_1| - |E_2|| > k$. Changing the color of one component from 1 to 2, we get a new coloring of the graph. For the new coloring, denote the set of edges colored i with $E'_i, i = 1, 2$. Since each component has less than k edges, $|E_1| > |E'_1| \geq k, |E'_2| \leq 2k - 2$. So $||E'_1| - |E'_2|| < ||E_1| - |E_2||$, a contradiction. Therefore we have $|E_1| \geq |E_2| \geq k$, so $G \in (2, k)$-LC. ◀

If G has at least two components, each with at least k edges, it is obviously a Yes-instance. Therefore by Lemma 13, we may assume there is exactly one component C with at least k edges in the graph. Denote the total number of edges in $G - V(C)$ with m'. Observe that if $m' \geq k$, trivially $G \in (2, k)$-LC. So assume that $m' < k$.

▶ **Lemma 14.** *If G is an irreducible graph for $(2, k)$-LC, $m' < k$ and $\Delta = \Delta(G) \geq 3k - 2m'$, then $G \in (2, k)$-LC.*

Proof. Let u be one of the vertices with degree Δ and $N(u)$ its neighbors. Because the graph is reduced by Reduction Rule $R_{1,k}$, u has at least $2k - 2m'$ neighbors which are not leaves. Arbitrarily select $k - m'$ vertices among them and for each one, select any neighbor but u. Color the selected vertices and $G - V(C)$ by 1. By construction, there are at least k edges colored 1 and there are at most $2k - 2m'$ colored vertices in $N(u)$. So there are at least k uncolored vertices in $N(u)$. We color them and u with 2. So $G \in (2, k)$-LC. ◀

▶ **Lemma 15.** *Let G be a graph with $\Delta < 3k$ and $|E(G)| \geq 8k$, then $G \in (2, k)$-LC.*

Proof. By Lemma 13, we may assume there exists a connected component C with at least k edges. In C, choose a minimal set $A \subseteq V(C)$ such that $|A| \leq k + 1$ and $|E(A)| = k + d \geq k$. We may find such a set A in the following way. Select arbitrarily a vertex in C and put it into A, then keep adding to this set some neighbor of some vertex in A until $|E(A)| = k + d \geq k$. Since each time we select a neighbor of A we strictly increase $|E(A)|$, $|A| \leq k + 1$. If there

is any vertex $u \in A$ with $|N_A(u)| \leq d$, then $A' = A \setminus \{u\}$ is a smaller vertex set such that $|E(A')| \geq k$. Thus, we may remove such vertices until $|E(A)| = k + d$ and for each vertex $u \in A$, $|N_A(u)| > d$. Denote $B = V(G) \setminus A$. We may assume $|E(B)| < k$, as otherwise $G \in (2, k)$-LC.

We now show that $|A| + d \leq k + 3$. Since every vertex $u \in A$ has $d_A(u) > d$, $|E(A)| = \frac{1}{2}\Sigma_{u \in A} d_A(u) \geq \frac{d+1}{2}|A|$. We have $k + d = |E(A)| \geq \frac{d+1}{2}|A|$, thus $|A| \leq \frac{2(k+d)}{d+1}$. Moreover as $d \leq |A| - 1$,

$$d + |A| \leq 2|A| - 1 \leq \frac{4(k+d)}{d+1} - 1 < \frac{4k}{d+1} + 3.$$

If $d \geq 3$, we are done; otherwise $d \leq 2$ and $d + |A| \leq 2 + k + 1 = k + 3$.

Let A_1, A_2, B_1, B_2 be a partition of $V(G)$ such that $A = A_1 \cup A_2$, $B = B_1 \cup B_2$, $|A_2| = 1$ and $|E(A, B_2)| < 2k$. Such a partition is possible: let $y = \text{argmax}\{|N_B(u)| : u \in A\}$ and initially take $A_1 = A \setminus \{y\}, A_2 = \{y\}, B_1 = B, B_2 = \emptyset$. Suppose $|E(A_1, B_1)| \leq k + |A_1|$. Then

$$\begin{aligned} |E(G)| &\leq |E(A)| + |E(B)| + |E(A_1, B_1)| + |E(A_2, B_1)| \\ &\leq (k + d) + (k - 1) + (k + |A| - 1) + \Delta \\ &\leq 7k + 1, \end{aligned}$$

a contradiction since $|E(G)| > 8k$. So, $|E(A_1, B_1)| > k + |A_1|$. We will consider two cases: $\max\{|N_{B_1}(u)| : u \in A\}$ is greater than k or not.

If so, observe that $|E(A_2, B_1)| = |E(\{y\}, B_1)| = \max\{|N_{B_1}(u)| : u \in A\} > k$. Move all vertices of $B_1 \setminus N(y)$ to B_2. We still have $|E(\{y\}, B_1)| > k$ and $|E(\{y\}, B_2)| = 0$. Moreover $B_1 \subseteq N(y)$. If $|E(A_1, B_2)| \geq k$, then G is in $(2, k)$-LC, thus $|E(A_1, B_2)| < k$. While $|E(\{y\}, B_1)| \geq k + 1$ and $|E(A_1, B_1)| \geq k + |A_1|$, move an arbitrary vertex from B_1 to B_2. After each move, $|E(\{y\}, B_1)| \geq k$ and $|E(A_1, B_1)| \geq k$, thus $|E(A_2, B_2)| < k$ and $|E(A_1, B_2)| < k$ as otherwise, G would be in $(2, k)$-LC.

Eventually, we have $|E(A_1, B_1)| < k + |A_1|$ or $|E(\{y\}, B_1)| = k$. Suppose $|E(A_1, B_1)| < k + |A_1|$. Then

$$\begin{aligned} |E(G)| &\leq |E(A)| + |E(B)| + |E(A_1, B_1)| + |E(A_1, B_2)| + |E(\{y\}, B)| \\ &\leq (k + d) + (k - 1) + (k + |A_1| - 1) + (k - 1) + \Delta \\ &\leq 4k - 3 + (d + |A|) + \Delta \\ &< 8k, \end{aligned}$$

a contradiction. Thus, $|E(A_1, B_1)| \geq k + |A_1|$ and $|E(\{y\}, B_1)| = k$. As $B_1 \subseteq N(y)$, we have $|B_1| = k$. We have found a new partition with the required properties and with $\max\{|N_{B_1}(u)| : u \in A\} \leq |B_1| = k$.

We now consider the case $\max\{|N_{B_1}(u)| : u \in A\} \leq k$. While there exists $u \in B_1$ such that $|E(A, B_2 \cup \{u\})| < 2k$, move u from B_1 to B_2. Then, (if and) while $|E(A_1, B_1)| \geq k + |A_1|$ and $|E(A_2, B_1)| < k + |A_2|$, move an arbitrary vertex from A_1 to A_2.

After all such moves, suppose that $|E(A_1, B_1)| < k + |A_1|$. If $|A_2| = 1$, we have $|E(A_2, B_1)| \leq \max\{|N_{B_1}(u)| : u \in A\} \leq k$, otherwise we moved some vertices from A_1 to A_2. Let u be the last one. Since $|E(A_2 \setminus \{u\}, B_1)| < k + |A_2 \setminus \{u\}|$, we know $|E(A_2, B_1)| \leq |E(A_2 \setminus \{u\}, B_1)| + \max\{|N_{B_1}(u)| : u \in A\} < k + |A_2| - 1 + k = 2k + |A_2| - 1$. For both

cases,

$$\begin{aligned} |E(G)| &= |E(A)| + |E(B)| + |E(A_1, B_1)| + |E(A_2, B_1)| + |E(A, B_2)| \\ &\leq (k+d) + (k-1) + (k + |A_1| - 1) + (2k + |A_2| - 2) + (2k-1) \\ &\leq 7k + d + |A| - 5 \\ &< 8k, \end{aligned}$$

which is impossible.

So, $|E(A_1, B_1)| \geq k + |A_1|$ which implies $|E(A_2, B_1)| \geq k + |A_2|$. For any vertex $u \in B_1$, we have $|E(A_1, B_1 \setminus \{u\})| \geq k$ and $|E(A_2, B_1 \setminus \{u\})| \geq k$ and we also obtain $|E(A, B_2 \cup \{u\})| \geq 2k$, i.e $E(A_1, B_2 \cup \{u\})$ or $E(A_2, B_2 \cup \{u\})$ has at least k edges. Thus, $G \in (2,k)$-LC. ◀

The lemmas of this section and the fact that their proofs can be turned into polynomial algorithms, imply the following:

▶ **Theorem 16.** *If G is irreducible for $(2,k)$-LC and has at least $8k$ edges, then $G \in (2,k)$-LC. Thus,* 2-Load Coloring *admits a kernel with less than $8k$ edges.*

Since we have a better bound for the number of edges in a kernel when $c = 2$, we may get a better approximation when $c = 2$.

▶ **Theorem 17.** *For every $\varepsilon > 0$, there is a $(4+\varepsilon)$-approximation algorithm for* 2-Max Load Coloring.

Proof. Let G be an instance for 2-Max Load Coloring with $m = 8p + q$ edges, where $0 \leq q < 8$. Let k_{opt} be the optimal solution of 2-Max Load Coloring on G, and observe that $k_{opt} \leq \lfloor \frac{m}{2} \rfloor \leq 4p + 3$. Let $p_0 = \lceil \frac{3}{\varepsilon} \rceil$. If $p \leq p_0 - 1$ then we can find k_{opt} in $O(1)$ time.

So assume that $p \geq p_0$. Note that $\frac{k_{opt}}{p} \leq \frac{4p+3}{p} \leq 4 + \varepsilon$. If G is irreducible for $(2,p)$-LC, $G \in (2,p)$-LC by Theorem 16, and so p gives the required approximation. We may assume that G is not irreducible for $(2,p)$-LC and reduce G to G'. If $|E(G')| \geq p$, then $G' \in (1,p)$-LC, and by Lemma 4, $G \in (2,p)$-LC. Again, p gives the required approximation.

Now assume that $|E(G')| < p$ and let $k'_{opt} = |E(G')|$ be the optimal solution of Max 1-Load Coloring on G'. Then $k'_{opt} + 1 \leq p$ and so an overload from $O_{1,p}$ is also an overload from $O_{1,k'_{opt}+1}$. Thus, G' can be derived from G using a reduction rule for $(2, k'_{opt} + 1)$-LC. Since $G' \notin (1, k'_{opt} + 1)$-LC, $G \notin (2, k'_{opt} + 1)$-LC. Thus $k_{opt} = k'_{opt} = |E(G')|$. So let our algorithm output $|E(G')|$ in this case. ◀

5 Discussions

To the best of our knowledge, we obtained the first nontrivial linear-edge kernel for a problem on general graphs. As we saw, such kernels can be used to obtain approximation algorithms. It would be interesting to obtain such kernels for other problems.

It is clear that our approximation algorithm is far from optimal; we will present an $O(c)$-approximation algorithm in the journal version of this paper.

Our linear-vertex kernel result implies an $O^*(c^{2ck})$-time algorithm for c-Load Coloring, which simply tests all the c-colorings of the kernel. However, it is possible that there is a subexponential FPT algorithm, since the problem No c-Load Coloring (the complement of c-Load Coloring) has small, but not constant, forbidden minors and is minor-bidimensional (see [5, 6] for more information on forbidden minors and bidimensionality).

Let $tw(G)$ denote the treewidth of G. The $O^*(2^{tw(G)})$-time algorithm for 2-Load Coloring from [9] can be generalized to an $O^*(c^{tw(G)})$-time algorithm for c-Load Coloring.

By [5, 6], if we require that the input G is H-minor-free for some fixed graph H, then we obtain an $O^*(c^{\sqrt{ck}})$-time algorithm. Unfortunately, there is no constant forbidden minor for No c-Load Coloring as membership in (c,k)-LC requires at least ck edges.

We think that there exists a subexponential algorithm for c-Load Coloring. By Theorem 4.12 of [5], and since branchwidth is linked to the treewidth up to a constant factor, any graph G contains an $(\Omega(\frac{tw(G)}{gen(G)}) \times \Omega(\frac{tw(G)}{gen(G)}))$-grid as a minor, where $gen(G)$ is the genus of G. Since the $(r \times r)$-grid is a forbidden minor for No c-Load Coloring for $r = \lceil \sqrt{(c+1)k} \rceil$, we have $tw(G) = O(\sqrt{ck}\ gen(G))$. Thus, we obtain an $O^*(c^{\sqrt{ck}\ gen(G)})$-time algorithm to solve c-Load Coloring, which is subexponential for graphs of bounded genus.

Acknowledgment. Research of GG was partially supported by Royal Society Wolfson Research Merit Award. Research of BS was supported by China Scholarship Council.

References

1. N. Ahuja, A. Baltz, B. Doerr, A. Prívetivý, and A. Srivastav. On the minimum load coloring problem. *J. Discrete Algorithms*, 5(3):533–545, 2007.
2. J.-P. Allouche and J. Shallit. The ring of k-regular sequences, II. *Theor. Comput. Sci.*, 307(1):3–29, 2003.
3. H. L. Bodlaender, F. V. Fomin, D. Lokshtanov, E. Penninkx, S. Saurabh, and D. M. Thilikos. (meta) kernelization. In *Foundations of Computer Science, FOCS 2009*, pages 629–638. IEEE Computer Society, 2009.
4. M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.
5. E. D. Demaine, F. V. Fomin, M. T. Hajiaghayi, and D. M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *J. ACM*, 52(6):866–893, 2005.
6. E. D. Demaine and M. T. Hajiaghayi. The bidimensionality theory and its algorithmic applications. *Comput. J.*, 51(3):292–302, 2008.
7. R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013.
8. F. V. Fomin, D. Lokshtanov, N. Misra, G. Philip, and S. Saurabh. Hitting forbidden minors: Approximation and kernelization. In *Symposium on Theoretical Aspects of Computer Science, STACS 2011*, volume 9 of *LIPIcs*, pages 189–200. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011.
9. G. Gutin and M. Jones. Parameterized algorithms for load coloring problem. *Inf. Process. Lett.*, 114(8):446–449, 2014.
10. S. Kratsch. Recent developments in kernelization: A survey. *Bulletin of the EATCS*, 113, 2014.
11. D. Lokshtanov, N. Misra, and S. Saurabh. Kernelization – preprocessing with a guarantee. In *The Multivariate Algorithmic Revolution and Beyond*, volume 7370 of *Lecture Notes in Computer Science*, pages 129–161. Springer, 2012.
12. E. Prieto. The method of extremal structure on the k-maximum cut problem. In *Theory of Computing 2005, Eleventh CATS 2005, Computing: The Australasian Theory Symposium, Newcastle, NSW, Australia, January/February 2005*, volume 41 of *CRPIT*, pages 119–126. Australian Computer Society, 2005.
13. J. Shallit, 2002. `http://oeis.org/A073121`.

Scheduling Two Competing Agents When One Agent Has Significantly Fewer Jobs

Danny Hermelin[1], Judith-Madeleine Kubitza[2], Dvir Shabtay[1], Nimrod Talmon[2], and Gerhard Woeginger[3]

1 **Ben Gurion University of the Negev, Israel**
hermelin@bgu.ac.il, dvirs@bgu.ac.il
2 **TU Berlin, Germany**
judith-madeleine.kubitza@campus.tu-berlin.de, nimrodtalmon77@gmail.com
3 **Eindhoven University of Technology, The Netherlands**
gwoegi@win.tue.nl

Abstract

We study a scheduling problem where two agents (each equipped with a private set of jobs) compete to perform their respective jobs on a common single machine. Each agent wants to keep the weighted sum of completion times of his jobs below a given (agent-dependent) bound. This problem is known to be NP-hard, even for quite restrictive settings of the problem parameters.

We consider parameterized versions of the problem where one of the agents has a small number of jobs (and where this small number constitutes the parameter). The problem becomes much more tangible in this case, and we present three positive algorithmic results for it. Our study is complemented by showing that the general problem is NP-complete even when one agent only has a single job.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases Parameterized Complexity, Multiagent Scheduling

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.55

1 Introduction

Scheduling is a well-studied research area which provides a fertile ground for combinatorial problems. In a typical scheduling problem, we are given a set of jobs that have to be scheduled on a set of machines, arranged according to a specific machine setting. The objective is to determine a schedule which minimizes a predefined scheduling criterion, such as the makespan, total weighted completion time, and total weighted tardiness. There are different machine settings such as a single machine, parallel machines, flow-shop and job-shop, and each scheduling problem may have different characteristics and constraints. We refer the reader, for example, to [22] for further examples of scheduling problems and for a detailed survey of classical results.

We consider a scheduling problem with two agents Alice and Bob who each own a set of jobs that are to be scheduled non-preemptively on a single machine. An instance of this scheduling problem consists of the following:

- two sets $\{J_1^A, \ldots, J_n^A\}$ and $\{J_1^B, \ldots, J_k^B\}$ of jobs;
- the processing times $p_1^A, \ldots, p_n^A$ and $p_1^B, \ldots, p_k^B$ of these jobs;
- the weights $w_1^A, \ldots, w_n^A$ and $w_1^B, \ldots, w_k^B$ of these jobs;
- two bounds A and B.

All processing times, all weights, and both bounds A and B are positive integers (notice that A and B are used both as labels and as integers). A *schedule* is simply a permutation of the union of both job sets, specifying the ordering in which the machine is executing the jobs. A schedule assigns to every job a corresponding *job completion time*; we denote by C_i^A and C_i^B respectively the completion times of the ith job of Alice and the ith job of Bob in some given schedule. The goal is to find a so-called *feasible schedule*, which satisfies the conditions

$$\sum_i w_i^A C_i^A \leq A \qquad \text{and} \qquad \sum_i w_i^B C_i^B \leq B.$$

Throughout the paper we refer to this problem as the Two Agent Scheduling problem.

The Two Agent Scheduling problem belongs to the family of *competitive multi-agent scheduling problems*: the jobs are partitioned into subsets, each belonging to a different agent that has his own scheduling criterion to optimize. Multi-agent scheduling problems were first introduced by Baker and Smith [3] and Agnetis et al. [2], which focus on the case with only two agents. For various combinations of the scheduling criteria, the objective in [3] is to minimize the weighted sum of the agents criteria. On the other hand, the objective in [2] is to minimize the scheduling criterion of the first agent, subject to a hard upper bound on the value of the criterion of the second agent. Following these two fundamental papers, numerous researchers have studied various combinations of multi-agent scheduling problems (see for instance Yuan et al. [23], Leung et al. [16], Lee et al. [14], Mor and Mosheiov [18], and Kovalyov et al. [13]). Detailed surveys of these problems appear in Perez-Gonzalez and Framinan [21] and in a recent book by Agnetis *et al.* [1].

Agnetis et al. [2] established that Two Agent Scheduling with unit weights is NP-complete. Furthermore, they designed a pseudo-polynomial time algorithm (which becomes polynomial time if all processing times are given in unary) for the problem. Lee et al. [14] extended this pseudo-polynomial time result to the case where the number of agents is an arbitrary but fixed constant (which is not part of the input). Oron et al. [20] proved that Two Agent Scheduling is NP-complete even when the processing times of all jobs are unit. They also show that the special case where Alice's jobs have unit weights is solvable in polynomial time.

1.1 Our contribution

It is natural to assume that the Two Agent Scheduling problem becomes more tangible if one of the agents has significantly fewer jobs than the other. Hence, we will parameterize the problem by the number k of jobs of agent Bob; throughout, we will tacitly assume that $k \ll n$. We will derive the following results:

1. The case where Alice's jobs have unit weights and the case where Alice's jobs have unit processing times are both fixed-parameter tractable with respect to parameter k. In other words, the problem can be solved in $f(k) \cdot n^{O(1)}$ time, where $f()$ is a function independent of n.
2. In stark contrast to the above positive results, the case where Alice's jobs have arbitrary weights and arbitrary processing times is NP-complete even if Bob has only a single job (that is, even if $k = 1$ holds).
3. If all job weights and processing times are given in unary and if Bob has a constant number of jobs, the problem is polynomial time solvable.

We also study another variant of Two Agent Scheduling where all jobs have unit weight: while Bob might have many jobs, we assume that his jobs have only a constant number t of different processing times. We show that this case is polynomial time solvable.

For obtaining the algorithms mentioned above, we will carefully analyze the combinatorial structure of Two Agent Scheduling. In the case where Alice's jobs have unit weights or unit processing times, it turns out that there is a very simple $n^{O(k)}$-time algorithm; however, it takes considerable work to improve this to a fixed-parameter algorithm. We model the problem in a non-trivial fashion as a mixed integer linear program (MILP), and then apply Lenstra's celebrated algorithm to it. This MILP modelling approach does not work for the general case where Alice's jobs have arbitrary weights and arbitrary processing times; in this general case we resort to dynamic programming to get an XP algorithm.

1.2 Further related work

The main contribution of this paper is a fixed-parameter algorithm for a natural scheduling problem. While scheduling theory and parameterized complexity (the study of fixed-parameter algorithms [8, 10, 19]) are two well-studied research fields, it seems that up to now the research on the interface between both areas has been rather limited. In fact, we found only a few research papers that study classical scheduling problems from the perspective of parameterized complexity. Bodlaender and Fellows [6] study the so-called precedence constrained k-processor scheduling problem. Fellows and McCartin [9] study the problem of scheduling unit length jobs on a single machine with precedence constraints. Both papers [6, 9] contain only hardness results. Some papers which include positive results on scheduling problems in the perspective of parameterized complexity are the paper by Halldórsson and Karlsson [12], the papers by van Bevern et al. [4, 5], and the paper by Mnich and Wiese [17], which shows that various classical scheduling problems are fixed-parameter tractable with respect to certain natural parameters.

2 Unit Weights and Unit Processing Times

In this section we show that Two Agent Scheduling is fixed-parameter tractable with respect to parameter k, the number of jobs that Bob has, in case Alice's jobs have either unit weights or unit processing times. We begin with the unit weight case.

2.1 Unit weights

▶ **Theorem 1.** Two Agent Scheduling *where Alice's jobs have unit weights is fixed-parameter tractable with respect to* k.

Agnetis *et al.* [2] observed that if an instance of Two Agent Scheduling where both agents have unit weights has a feasible schedule, then there is always a feasible schedule where the relative order amongst the jobs of each agent is according to the SPT (Shortest Processing Time first) rule. That is, we can assume that each job of each agent proceeds all jobs with greater processing times of the same agent in a feasible schedule. This remains true for Alice's jobs even if Bob's jobs have arbitrary weights. Thus, we assume $J_1^A, \ldots, J_n^A$ are already indexed according to the SPT order (that is, that $p_i^A \leq p_{i+1}^A$ for all $i \in \{1, \ldots, n-1\}$). Since Bob has only k jobs, we can iterate through all relative orderings of his jobs and "guess" his relative order. Thus, by allowing an additional multiplicative factor of $k!$ to the running time of our algorithm, we can assume that we know this ordering, and that $J_1^B, \ldots, J_k^B$ are already sorted accordingly.

Therefore, to determine whether a feasible schedule is actually possible, we only need to figure out if it is possible to interleave the two ordered sets of jobs together in a way that satisfies both Alice's and Bob's bounds on their total weighted completion times. Towards

this aim, we formalize the problem as a mixed integer linear program (MILP) where the number of integer variables is k. We complete the proof by using the celebrated result of Lenstra [15] which states that determining whether a given MILP has a feasible solution is fixed-parameter tractable with respect to the number of integer variables.

Alice's total completion time bound

For each job J_i^B, define an integer variable x_i representing the number of jobs belonging to Alice that are scheduled before J_i^B. For each $1 \leq i \leq k$, we add a pair of constraints ensuring that $0 \leq x_i \leq n$, and for $i \neq n$, we add the constraint $x_i \leq x_{i+1}$. Using the variables x_i, we encode the bound on the total completion time of Alice's jobs by adding the linear constraint

$$\left(\sum_{i=1}^{n}(n-i+1)\cdot p_i^A\right) + \left(\sum_{i=1}^{k}(n-x_i)\cdot p_i^B\right) \leq A. \tag{1}$$

▶ **Lemma 2.** *Assuming the value of each variable x_i equals the number of Alice's jobs that are scheduled before J_i^B, the left-hand side of constraint (1) is precisely the total completion time of Alice's jobs.*

Proof. Observe that the first term in the left-hand side of constraint (1) is precisely the sum of completion times of Alice's jobs when no job of Bob is scheduled at all. We now add Bob's jobs according to the intended meaning of the variables $x_1, \ldots, x_k$. If there are x_i jobs of Alice prior to J_i^B in the presumed schedule, then J_i^B causes an increase of p_i^B to the completion time of $n - x_i$ jobs of Alice. Thus, adding all of Bob's jobs causes an additional increase of $\sum_{i=1}^{k}(n-x_i)\cdot p_i^B$ to the total completion time of Alice's jobs. ◀

Bob's total completion time bound

The encoding of Bob's bound is a bit more involved. Specifically, for each J_i^B, we introduce a real-valued variable y_i which we would like to be equal to the contribution of Alice's jobs to the completion time of J_i^B. Note that by our intended meaning for variable x_i, this is precisely $\sum_{j=1}^{x_i} p_j^A$. However, we cannot encode this directly as a linear constraint. We therefore introduce n additional real-valued variables corresponding to J_i^B, denoted as $y_{i,j}$ for $j \in \{1, \ldots, n\}$, which are ensured to be non-negative by adding constraints $y_{i,j} \geq 0$ for each j. The $y_{i,j}$ variables are used to provide upper-bounds to the "steps" in the contribution of Alice's jobs as depicted in Fig. 1. Accordingly, we add the constraints $y_{i,j} \geq (x_i - j + 1)\cdot(p_j^A - p_{j-1}^A)$, for each $j \in \{1, \ldots, n\}$. (Naturally, we set here $p_0^A = 0$.) Furthermore, we add the constraint $y_i \geq \sum_{j=1}^{n} y_{i,j}$ so that y_i will equal its intended meaning.

Finally, to complete the construction of our MILP, we add the following constraint which encodes the bound on Bob's total weighted completion time.

$$\left(\sum_{i=1}^{k} w_i^B \cdot \sum_{j=1}^{i} p_j^B\right) + \left(\sum_{i=1}^{k} w_i^B \cdot y_i\right) \;\leq\; B. \tag{2}$$

▶ **Lemma 3.** *Assuming the value of each variable x_i equals the number of Alice's jobs that are scheduled before J_i^B, the left-hand side of constraint (2) is an upper-bound to the total completion time of Bob's jobs in any feasible solution.*

Proof. Observe that the first term in the left-hand side of constraint (2) is the total weighted completion time of Bob's jobs ordered $J_1^B, \ldots, J_k^B$, assuming no job of Alice has been

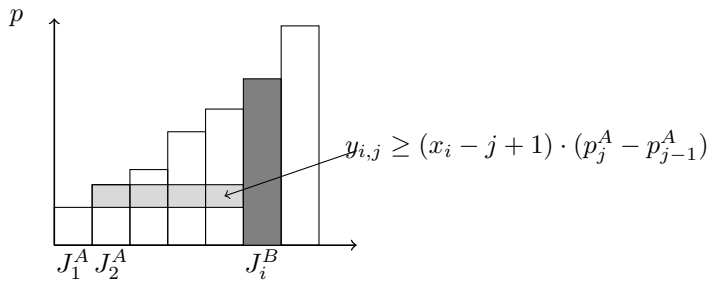


Figure 1 The contribution of x_i jobs that belong to Alice which are scheduled prior to J_i^B increases the completion time of this job by $\sum_{j=1}^{x_i} p_j^A$. The variable $y_{i,j}$ is intended to capture j'th "step" of this contribution.

scheduled. We argue that the second term upper-bounds the contribution of Alice's jobs. For this, it suffices to show that the contribution of Alice's jobs to the completion time of J_i^B, for each $i \in \{1, \ldots, k\}$, is at most y_i. We know that this contribution is $\sum_{j=1}^{x_i} p_j^A$, assuming x_i is the number of Alice's jobs that are scheduled prior to J_i^B. Since we are concerned only with feasible solutions where the constraints on the variables $y_i, y_{i,1} \ldots, y_{i,n}$ are met, we have that the following hold, and we are done:

$$y_i \geq \sum_{j=1}^{n} y_{i,j} \geq \sum_{j=1}^{n} \max\{0, (x_i - j + 1) \cdot (p_j^A - p_{j-1}^A)\} \geq \sum_{j=1}^{x_i} (x_i - j + 1) \cdot (p_j^A - p_{j-1}^A)$$
$$= x_i p_1^A + (x_i - 1)(p_2^A - p_1^A) + \cdots + (p_{x_i}^A - p_{x_i-1}^A) = \sum_{j=1}^{x_i} p_j^A. \quad \blacktriangleleft$$

To summarize, combining Lemma 2 and Lemma 3 shows that, assuming there exists a feasible schedule where Bob's jobs are scheduled according to our assumed order, the MILP described above will have a feasible solution, and otherwise no such solution exists. Moreover, the MILP described has only k integer variables. Thus, iterating through all possible orderings of Bob's jobs, and using Lenstra's algorithm to determine whether the MILP corresponding to each order has a feasible solution, gives us the fixed-parameter algorithm promised in Theorem 1.

2.2 Unit processing times

The somewhat symmetric problem, where Alice's jobs have arbitrary weights and unit processing times can also be shown to be fixed-parameter tractable with respect to k using similar ideas as above. The first crucial observation in this case, which can be seen by a simple exchange argument, is that we can assume that in any feasible schedule Alice's jobs are sorted amongst themselves in a non-increasing weight order. Thus, assuming $w_1^A \geq \cdots \geq w_n^A$, we know that J_i^A will be scheduled before J_{i+1}^A for all $i \in \{1, \ldots, n-1\}$. As in the proof of Theorem 1, by guessing the relative order of Bob's job in the schedule, we can assume that the same applies for Bob's jobs as well. Again we introduce variables $x_1, \ldots, x_k$, but this time x_i represents the number of Alice's jobs to be scheduled after J_i^B. Similarly to before, we add all constraints $0 \leq x_i \leq n$ and $x_i \geq x_{i+1}$. The bound on the weighted sum of completion times of Bob's jobs can be expressed as

$$\left(\sum_{i=1}^{k} w_i^B \sum_{j=1}^{i} p_j^B \right) + \left(\sum_{i=1}^{k} w_i^B (n - x_i) \right) \leq B,$$

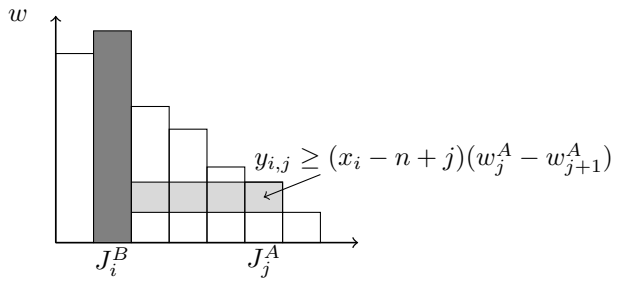


Figure 2 Inserting job J_i^B increases the weighted completion time of each following job J_j^A by $p_i^B w_j^A$. Thus, the contribution of J_i^B to the total weighted completion time of Alice's jobs is $p_i^B \sum_{j=n-x_i+1}^{n} w_j^A$. The variables $y_{i,j}$ are used to lower-bound the steps of the sum $\sum_{j=n-x_i+1}^{n} w_j^A$.

where the first term in the left-hand side is the contribution of Bob's jobs, and the second term in the left-hand side is the contribution of Alice's jobs.

For the total weighted completion time of Alice's jobs, we again introduce a set of real-valued variables $y_i, y_{i,1}, \ldots, y_{i,n}$ for each $i \in \{1 \ldots, k\}$. Here, variable y_i is meant to encode the contribution of J_i^B to the total weighted completion time of Alice's jobs. Note that this is precisely $p_i^B \sum_{j=n-x_i+1}^{n} w_j^A$ (see Fig. 2). We use the variables $y_{i,j}$ to encode lower-bounds on the steps of the sum $\sum_{j=n-x_i+1}^{n} w_j^A$, as done in the proof of Theorem 1, by adding the constraints $y_{i,j} \geq (x_i - n + j)(w_j^A - w_{j+1}^A)$ and $y_{i,j} \geq 0$ for all $j \in \{1, \ldots, n\}$. (Naturally we set $w_{n+1}^A = 0$.) Finally, we encode the bound on Alice's total weighted completion time by

$$\left(\sum_{i=1}^{n} i w_i^A\right) + \left(\sum_{i=1}^{k} p_i^B y_i\right) \leq A.$$

▶ **Theorem 4.** Two Agent Scheduling *where Alice's jobs have unit processing time is fixed-parameter tractable with respect to* k.

3 Single Job Bob

In the previous section we showed that when Alice's jobs have either unit weights or unit processing times, Two Agent Scheduling becomes fixed-parameter tractable in the number of Bob's jobs. In this section we complement this result by showing that when Alice's jobs have arbitrary weights and processing times, the problem becomes NP-complete already when Bob has a single job. Note that this rules out a similar fixed-parameter algorithm for general Two Agent Scheduling in the strongest possible sense (indeed, even an XP-algorithm cannot exist); such an algorithm would imply P=NP.

▶ **Theorem 5.** Two Agent Scheduling *where Bob has a single job is* NP*-complete.*

Proof. We provide a reduction from the NP-complete Partition problem [11]: Given a set $X = \{x_1, \ldots, x_n\}$ of positive integers (encoded in binary) with $\sum_{i=1}^{n} x_i = 2Z$, determine whether X can be partitioned into two sets S_1 and S_2 such that $\sum_{x_i \in S_1} x_i = \sum_{x_i \in S_2} x_i = Z$.

Given an instance X to the Partition problem, we create for each element x_i a job J_i^A for Alice with both processing time and weight equal to x_i. This gives n jobs for Alice with $p_i^A = w_i^A = x_i$ for all $i \in \{1, \ldots, n\}$. For Bob, we create a single job with both unit processing time and unit weight. Thus, $p_1^B := w_1^B := 1$. We set the bound A on the total weighted completion time of Alice's jobs to be $A = \sum_{i=1}^{n} \sum_{j=1}^{i} x_i x_j \; + \; Z$. We set the bound B for

Bob to be $B = Z + 1$. This completes our construction of the Two Agent Scheduling instance.

Suppose that X can be partitioned into two sets S_1 and S_2 with $\sum_{x_i \in S_1} x_i = \sum_{x_i \in S_2} x_i = Z$. We create a schedule σ where we first schedule all of Alice's jobs corresponding to elements of S_1 in an arbitrary order, followed by Bob's job, followed by all of Alice's jobs corresponding to the elements of S_2 in an arbitrary order. Observe that Bob's job completes in σ after $\sum_{x_i \in S_1} x_i + 1 = Z + 1$ time units, and so his total weighted completion time bound is met. To see that Alice's bound is met, observe that without Bob's job the total weighted completion time of Alice's jobs in σ is precisely $\sum_{i=1}^{n} \sum_{j=1}^{i} x_i x_j$. Adding Bob's job increases the competition time of Alice's jobs that correspond to elements of S_2 by a unit. Thus, it contributes precisely $\sum_{x_i \in S_2} x_i = Z$ to the total weighted completion time of Alice, and so Alice's bound is met as well.

For the other direction, suppose there is a feasible schedule σ to our Two Agent Scheduling problem. Let $\mathcal{J}_1$ denote the set of Alice's jobs that are placed before Bob's job in σ, and let $\mathcal{J}_2$ denote her remaining jobs. Since Bob's bound is satisfied in σ, it must be that $\sum_{J_i^A \in \mathcal{J}_1} p_i^A \leq B - p_1^B = Z$. Moreover, note that the total weighted completion time of Alice is $\sum_{i=1}^{n} \sum_{j=1}^{i} x_i x_j + \sum_{J_i^A \in \mathcal{J}_2} p_i^A$. Thus, since Alice's bound is also met by σ, it must be that $\sum_{J_i^A \in \mathcal{J}_2} p_i^A \leq Z$. Since the sum of all processing times of Alice's jobs is $2Z$, we get that $\sum_{J_i^A \in \mathcal{J}_1} p_i^A = \sum_{J_i^A \in \mathcal{J}_2} p_i^A = Z$, and so $S_1 = \{x_i : J_i^A \in \mathcal{J}_1\}$ and $S_2 = \{x_i : J_i^A \in \mathcal{J}_2\}$ is a solution to our Partition instance. ◀

4 Unary Encoded Weights and Processing Times

Agnetis *et al.* [1] showed that Two Agent Scheduling is strongly NP-hard, even when the input is given in unary. The unit-weight variant of Two Agent Scheduling, however, can be solved by a dynamic programming that is based on the SPT ordering of the jobs of both agents [2]. Roughly speaking, the algorithm computes a table $T[\cdot, \cdot, \cdot]$, where the entry $T[C, i, j]$ stores the minimum total completion time of Bob's jobs when $\{J_1^A, \ldots, J_i^A\} \cup \{J_1^B, \ldots, J_j^B\}$ are scheduled together and Alice's total completion time is at most C. Note that this works because the SPT order of Alice's and Bob's jobs are preserved.

When jobs have arbitrary weights the SPT rule no longer applies, and we do not know the relative order of the jobs of each agent in advance. Nevertheless, we can easily extend the algorithm above to an algorithm which is fixed-parameter in k when only Bob's jobs have arbitrary weights, by guessing the relative order of his jobs. For unary encoding, this gives a faster algorithm than the algorithm proposed in Theorem 1. However, when the jobs of both Alice and Bob have arbitrary weights, this strategy fails. In the remainder of the section we present a more elaborate dynamic program that will give the following:

▶ **Theorem 6.** Two Agent Scheduling *can be solved in* $O(n \cdot k! \cdot B(W_A P_A)^{k+1})$ *time, where* $W_A = \sum_{i=1}^{n} w_i^A$ *and* $P_A = \sum_{i=1}^{n} p_i^A$. *The algorithm is polynomial if Alice's jobs' weights and processing times, and Bob's bound are given in unary, and* k *is a fixed constant.*

In order to derive the result in Theorem 6 above, we start, as in Section 2, by fixing the sequence in which Bob's jobs are scheduled. We then renumber Bob's jobs according to this sequence such that J_i^B is scheduled before J_{i+1}^B for $i = 1, ..., k-1$, and are left with the following *feasibility subproblem*: can Alice's jobs be sequenced and the two ordered sets of jobs be interleaved such that the total weighted completion time of both Alice and Bob will not exceed the bounds A and B, respectively? Clearly, there is a feasible schedule to

our Two Agent Scheduling problem if and only if at least one of our $k!$ instances of the feasibility subproblem has a solution.

Let A_i be the set of Alice's jobs that are scheduled between jobs J_i^B and J_{i+1}^B for $i \in \{1, ..., k-1\}$, and let A_0 and A_k be the set of Alice's jobs that are scheduled before job J_1^B and after job J_k^B, respectively. The following lemma, easily proven by a simple pair-wise interchange argument, helps us to partially answer the sequencing part of the feasibility subproblem.

▶ **Lemma 7.** *If the answer to the feasibility problem is yes, then there is a schedule that yields a yes-answer where the jobs in each A_i are scheduled in a non-decreasing order of their processing time by weight ratio (that is, according to the* WSPT *rule).*

We next use the above lemma to construct a dynamic programming algorithm to answer the feasibility subproblem in $O(n \cdot B(W_A P_A)^{k+1})$ time. We start by renumbering Alice's jobs in a non-increasing order of p_i^A/w_i^A, such that $p_i^A/w_i^A \geq p_{i+1}^A/w_{i+1}^A$ for $i = 1, \ldots, n-1$.

Now, let $F_j[C, W_0, ..., W_k, P_0, ..., P_k]$ be the minimum weighted sum of completion time of Alice's jobs in a partial schedule that includes jobs $J_1^A, ..., J_j^A$ where W_i and P_i represents the total weight and processing time of Alice's jobs that are assigned to A_i, and C corresponds to the weighted sum of completion times of Bob's jobs.

Initially, we have that none of Alice's jobs are scheduled. Thus, $A_i = \emptyset$ for $i = 0, \ldots, k$, and the total weighted completion time of Bob's jobs is given by $C = \Sigma_{i=1}^k w_i^B \Sigma_{j=1}^i p_j^B$. Thus, the initial condition for our recursion is:

$$F_0[C, W_0, ..., W_k, P_0, ..., P_k] = \begin{cases} 0, & \text{if } C = \Sigma_{i=1}^k w_i^B \Sigma_{j=1}^i p_j^B \\ & \text{and } W_i = P_i = 0 \text{ for all } i, \\ \infty, & \text{otherwise.} \end{cases} \tag{3}$$

Consider now a state value $F_j[C, W_0, ..., W_k, P_0, ..., P_k]$ and assume that job J_j^A is assigned to A_i. Following Lemma 7, job J_j^A is scheduled first in A_i, which leads to an increase of p_j^A units of time in the completion time of Alice's jobs that have already been assigned to sets $A_i, A_{i+1}, ..., A_k$, and in the completion time of all of Bob's jobs J_j^B for $j = i+1, \ldots, k$. Given that J_j^A is completed at time $\Sigma_{l=0}^{i-1}(P_l + p_l^B) + p_j^A$, we obtain the following recursion:

$$\begin{aligned} F_j[C, W_0, \ldots, W_k, P_0, ..., P_k] = \\ \min_{i=0,\ldots,k} \{ & F_{j-1}[C - p_j^A \Sigma_{l=i+1}^k w_l^B, W_0', \ldots, W_k', P_0', ..., P_k'] \\ & + p_j^A \Sigma_{l=i}^k W_l + w_j^A \Sigma_{l=0}^{i-1}(P_l + p_l^B) \}, \end{aligned} \tag{4}$$

where $W_l' = W_l - w_j^A$ and $P_l' = P_l - p_j^A$ if $l = i$; and $W_l' = W_l$ and $P_l' = P_l$, otherwise.

Starting from the initial condition in (3), we compute $F_j[C, W_0, ...W_k, P_0, ..., P_k]$ by using (4), for any $j \in \{1, ..., n\}$, $W_i \in \{0, 1, \ldots, \Sigma_{l=1}^j w_l^A\}$, $P_i \in \{0, \ldots, \Sigma_{l=1}^j p_l^A)$, and $C \leq B$, where $i \in \{0, \ldots, k\}$. At the end of the dynamic programming procedure, we output a yes-answer for the feasibility subproblem if there exists a computed state with $F_n[C, W_0, ...W_k, P_0, \ldots, P_k] \leq A$. Otherwise, we output a no-answer.

Given that $W_i \in [0, \Sigma_{l=1}^j w_l^A]$, $W_i \in [0, \Sigma_{l=1}^j p_l^A]$, for $i = 0, \ldots, k$, and $C \leq B$, it follows that our dynamic program algorithm runs in $O(n \cdot B(W_A P_A)^{k+1})$ time. The fact that there are only $k!$ possible ways to sequence Bob's jobs yields the result in Theorem 6.

5 Bob Has Only a Few Job Types

In this section, we do not restrict the number k of Bob's jobs, but the number t of types for his jobs. Specifically, we consider Two Agent Scheduling when all jobs are of unit

weight and there is only a small number t of different processing times for Bob's jobs. Let B_i be the set of Bob's jobs of type i, that is, the set of Bob's jobs of the ith processing time, for $1 \leq i \leq t$.

We say that schedule σ_1 *dominates* schedule σ_2, if the sums of completion times for both Alice and Bob under σ_1 are smaller or equal to the sums of completion times for them under σ_2. A *Pareto-optimal point* (or, *Pareto-optimal schedule*) is a schedule which is not dominated by any other schedule. Crucially, if there is a feasible schedule to an instance of Two Agent Scheduling, then there is a feasible schedule to this instance which is a Pareto-optimal schedule.

The next lemma will be used to construct a polynomial-time algorithm for Two Agent Scheduling when all jobs are of unit weights and when t is a constant, based on iterating over a representative set of Pareto-optimal points.

We say that job J_j^A interleaves with B_i in schedule σ, if σ includes a subschedule of type $\{b_1, J_j^A, b_2\}$, where $b_1, b_2 \in B_i$ are non-empty sets (that is, the job J_j^A is scheduled before each job of b_2 and after each job of b_1).

▶ **Lemma 8.** *For any Pareto-optimal point, there exists a Pareto-optimal schedule in which, for each $1 \leq i \leq t$, at most a single job of Alice interleaves with each B_i.*

Proof. Consider a Pareto-optimal schedule σ that includes, for some $1 \leq i \leq t$, at least two jobs of Alice, J_j^A and J_{j+1}^A, both interleaving with B_i. Accordingly, σ includes a subschedule $\{b_1, J_j^A, b_2, J_{j+1}^A, b_3\}$, where $b_1, b_2, b_3 \in B_i$ are non-empty sets. Construct an alternative schedule σ' as follows: move J_j^A, $\min\{|b_1|, |b_3|\}$ positions to the left and J_{j+1}^A, $\min\{|b_1|, |b_3|\}$ positions to the right. Since all jobs in B_i have the same processing time, the sum of completion times of Alice's jobs remains the same. Moreover, the sum of completion times of Bob's jobs decreases by $\min\{|b_1|, |b_3|\} \cdot \left(p_{j+1}^A - p_j^A\right)$. The lemma now follows from the fact that the jobs are numbered according to the SPT rule and that in σ' at least one job out of the pair $\{J_j^A, J_{j+1}^A\}$ does not interleave with B_i. ◀

Following Lemma 8 above, we can construct a set of Pareto-optimal points which represents all Pareto-optimal points (in the sense that we have a representative for each equivalence class of the Pareto-optimal points, where two Pareto-optimal points are in the same equivalence class if the sums of completion times for both Alice and Bob are the same in both schedules) by constructing the entire set of schedules in which, for each $1 \leq i \leq t$, at most a single job of Alice interleaves with each B_i. We call any schedule of this type a *normal* schedule, and note that any such schedule can be concisely described as: "$a_1 b_1 a_{1^*} b_{1^*} a_2 b_2 a_{2^*} b_{2^*} \cdots a_t b_t a_{t^*} b_{t^*} a_{\text{rest}}$", where $a_{\text{rest}} + \sum_{i=1}^{t}(a_i + a_{i^*}) = n$, $0 \leq a_{i^*} \leq 1$ for each $1 \leq i \leq t$, and $b_i + b_{i^*} = |B_i|$ for each $1 \leq i \leq t$, with the intended meaning that a_1 of Alice's jobs are scheduled first, then b_1 of Bob's jobs of the first type, then a_{1^*} of Alice's jobs, then b_{1^*} of Bob's jobs of the first type, then a_2 of Alice's jobs, then b_2 of Bob's jobs of the second type, then a_{2^*} of Alice's jobs, then b_{2^*} of Bob's jobs of the second type, and so on, until, finally, a_{rest} of Alice's jobs are scheduled.

The number of normal schedules is upper-bounded by $k^t 2^t n^t$, since they are uniquely defined by all possible values of b_i, all possible (binary) values of a_{i^*}, and all possible values of a_i, for $1 \leq i \leq t$. Polynomial-time algorithm then follows by iteratively checking the feasibility all possible normal schedules.

6 Discussion

We would like to point out several directions for future research.

- While we determined the (parameterized) complexity of Two Agent Scheduling when the input is given in binary, our understanding of the complexity Two Agent Scheduling when the input is given in unary is lacking.
- It is natural to study Two Agent Scheduling when considering other objective functions. Similarly, it also makes sense to consider other, related, scheduling problems.
- Finally, we believe that our MILP formulation and the ideas underlying it, as described in the proof of Theorem 1, might be useful for other problems. This is done, to some extent, in [7, Theorem 1 and Theorem 2], and we would like to see other problems which have a similar structure that might be utilized in similar ways.

References

1 Alessandro Agnetis, Jean-Charles Billaut, Stanisław Gawiejnowicz, Dario Pacciarelli, and Ameur Soukhal. *Multiagent Scheduling: Models and Algorithms.* Springer, 2014.

2 Allesandro Agnetis, Pitu B. Mirchandani, Dario Pacciarelli, and Andrea Pacifici. Scheduling problems with two competing agents. *Operations Research*, 52(2):229–242, 2004.

3 Kenneth R. Baker and J. Cole Smith. A multiple-criterion model for machine scheduling. *Journal of Scheduling*, 6(1):7–16, 2003.

4 René van Bevern, Matthias Mnich, Rolf Niedermeier, and Mathias Weller. Interval scheduling and colorful independent sets. In *Proceedings of the 23th International Symposium on Algorithms and Computation (ISAAC'12)*, volume 7676 of *LNCS*, pages 247–256. Springer, 2012.

5 René van Bevern, Rolf Niedermeier, and Ondrej Suchý. A parameterized complexity view on non-preemptively scheduling interval-constrained jobs: few machines, small looseness, and small slack. *arXiv preprint arXiv:1005.4159*, 2015.

6 Hans L. Bodlaender and Michael R. Fellows. W[2]-hardness of precedence constrained k-processor scheduling. *Operations Research Letters*, 18(2):93–97, 1995.

7 Robert Bredereck, Piotr Faliszewski, Rolf Niedermeier, Piotr Skowron, and Nimrod Talmon. Elections with few candidates: Prices, weights, and covering problems. In *Proceedings of the 4th International Conference on Algorithmic Decision Theory (ADT'15)*, pages 414–431. Springer, 2015.

8 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity.* Springer, 2013.

9 Michael R. Fellows and Catherine McCartin. On the parametric complexity of schedules to minimize tardy tasks. *Theoretical computer science*, 298(2):317–324, 2003.

10 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory.* Springer, 2006.

11 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, 1979.

12 Magnús M. Halldórsson and Ragnar K. Karlsson. Strip graphs: Recognition and scheduling. In *Proceedings of the 32th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'06)*, volume 4271 of *LNCS*, pages 137–146. Springer, 2006.

13 Mikhail Y. Kovalyov, Ammar Oulamara, and Ameur Soukhal. Two-agent scheduling on an unbounded serial batching machine. In *Proceedings of the Second International Symposium on Combinatorial Optimization (ISCO'12)*, volume 3787 of *LNCS*, pages 427–438. Springer, 2012.

14 Kangbok Lee, Byung-Cheon Choi, Joseph Y.-T. Leung, and Michael L. Pinedo. Approximation algorithms for multi-agent scheduling to minimize total weighted completion time. *Information Processing Letters*, 109(16):913–917, 2009.

15 Hendrik W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983.

16 Joseph Y.-T. Leung, Michael Pinedo, and Guohua Wan. Competitive two-agent scheduling and its applications. *Operations Research*, 58(2):458–469, 2010.

17 Matthias Mnich and Andreas Wiese. Scheduling and fixed-parameter tractability. In *Proceedings of the 17th International Conference on Integer Programming and Combinatorial Optimizationon (IPCO'14)*, volume 8494 of *LNCS*, pages 381–392. Springer, 2014.

18 Baruch Mor and Gur Mosheiov. Single machine batch scheduling with two competing agents to minimize total flowtime. *European Journal of Operational Research*, 215(3):524–531, 2011.

19 Rolf Niedermeier. Invitation to fixed-parameter algorithms. Habilitation thesis, Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, September 2002.

20 Daniel Oron, Dvir Shabtay, and George Steiner. Single machine scheduling with two competing agents and equal job processing times. *European Journal of Operational Research*, 2015.

21 Paz Perez-Gonzalez and Jose M. Framinan. A common framework and taxonomy for multicriteria scheduling problems with interfering and competing jobs: Multi-agent scheduling problems. *European Journal of Operational Research*, 235(1):1–16, 2014.

22 Michael L. Pinedo. *Scheduling: theory, algorithms, and systems*. Springer Science & Business Media, 2012.

23 Jinjiang Yuan, Weiping Shang, and Qi Feng. A note on the scheduling with two families of jobs. *Journal of Scheduling*, 8(6):537–542, 2005.

On the Workflow Satisfiability Problem with Class-independent Constraints

Jason Crampton, Andrei Gagarin, Gregory Gutin, and Mark Jones

Royal Holloway, University of London, Egham, Surrey TW20 0EX, UK

Abstract

A workflow specification defines sets of steps and users. An authorization policy determines for each user a subset of steps the user is allowed to perform. Other security requirements, such as separation-of-duty, impose constraints on which subsets of users may perform certain subsets of steps. The *workflow satisfiability problem* (WSP) is the problem of determining whether there exists an assignment of users to workflow steps that satisfies all such authorizations and constraints. An algorithm for solving WSP is important, both as a static analysis tool for workflow specifications, and for the construction of run-time reference monitors for workflow management systems. Given the computational difficulty of WSP, it is important, particularly for the second application, that such algorithms are as efficient as possible.

We introduce class-independent constraints, enabling us to model scenarios where the set of users is partitioned into groups, and the identities of the user groups are irrelevant to the satisfaction of the constraint. We prove that solving WSP is fixed-parameter tractable (FPT) for this class of constraints and develop an FPT algorithm that is useful in practice. We compare the performance of the FPT algorithm with that of SAT4J (a pseudo-Boolean SAT solver) in computational experiments, which show that our algorithm significantly outperforms SAT4J for many instances of WSP. User-independent constraints, a large class of constraints including many practical ones, are a special case of class-independent constraints for which WSP was proved to be FPT (Cohen *et al.*, J. Artif. Intel. Res. 2014). Thus our results considerably extend our knowledge of the fixed-parameter tractability of WSP.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Workflow Satisfiability Problem, Constraint Satisfaction Problem, fixed-parameter tractability, user-independent constraints

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.66

1 Introduction

It is increasingly common for organizations to computerize their business and management processes. The co-ordination of the tasks or steps that comprise a computerized business process is managed by a workflow management system (or business process management system). Typically, the execution of these steps will be triggered by a human user, or a software agent acting under the control of a human user, and each step may only be executed by an *authorized* user. Thus a workflow specification will include an authorization policy defining which users are authorized to perform which steps.

In addition, many workflows require controls on the users that perform certain sets of steps [1, 2, 3, 7, 14]. Consider a simple purchase-order system in which there are four steps: raise-order (s_1), acknowledge-receipt-of-goods (s_2), raise-invoice (s_3), and send-payment (s_4). The workflow specification for the purchase-order system includes rules to prevent fraudulent use of the system, the rules taking the form of *constraints* on users that can perform pairs of steps in the workflow: the same user may not raise the invoice (s_3) and sign for the

10th International Symposium on Parameterized and Exact Computation (IPEC 2015).
Editors: Thore Husfeldt and Iyad Kanj; pp. 66–77

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

goods (s_2), for example. Such a constraint is known as a *user-independent* (UI) constraint, since the specific identities of the users that perform these steps are not important, only the relationship between them (in this example, the identities must be different).

Once we introduce constraints on the execution of workflow steps, it may be impossible to find a *valid plan* – an assignment of authorized users to workflow steps such that all constraints are satisfied. The Workflow Satisfiability Problem (WSP) takes a workflow specification as input and outputs a valid plan if one exists. WSP is known to be NP-hard, even when the set of constraints only includes constraints having a relatively simple structure (and arising regularly in practice). In particular, the Graph k-Colorability problem can be reduced to a special case of WSP in which the workflow specification only includes separation-of-duty constraints [14]. Clearly, it is important to be able to determine whether a workflow specification is satisfiable at design time. Equally, when users select steps to execute in a workflow instance, it is essential that the access control mechanism can determine whether (a) the user is authorized, (b) allowing the user to execute the step would render the instance unsatisfiable. Thus, the access control mechanism must incorporate an algorithm to solve WSP, and that algorithm needs to be as efficient as possible.

Wang and Li [14] observed that, in practice, the number k of steps in a workflow will be small, relative to the size of the input to WSP; specifically, the number of users is likely to be an order of magnitude greater than the number of steps. This observation led them to set k as the parameter and to study the problem using tools from parameterized complexity. In doing so, they proved that the problem is *fixed-parameter tractable* (FPT) for simple classes of constraints. However, Wang and Li also showed that for many types of constraints the problem is fixed-parameter *intractable* (unless FPT $\neq$ W[1] is false). Hence, it is important to be able to identify those types of practical constraints for which WSP is FPT.

Recent research has made significant progress in understanding the fixed-parameter tractability of WSP. In particular, Cohen *et al.* [5] introduced the notion of patterns and, using it, proved that WSP is FPT (irrespective of the authorization policy) if all constraints in the specification are UI. This result is significant because most constraints in the literature – including separation-of-duty, cardinality and counting constraints – are UI [5]. Using a modified pattern approach, Karapetyan *et al.* [11] provided both a short proof that WSP with only UI constraints is FPT and a very efficient algorithm for WSP with UI constraints.

However, it is known that not all constraints that may be useful in practice are UI. Consider a situation where the set of users is partitioned into groups (such as departments or teams) and we wish to define constraints on the groups, rather than users. In our purchase order example, suppose each user belongs to a specific department. Then it would be reasonable to require that steps s_1 and s_2 are performed by different users belonging to the same department. There is little work in the literature on constraints of this form, although prior work has recognized that such constraints are likely to be important in practice [7, 14], and it has been shown that such constraints present additional difficulties when incorporated into WSP [9].

In this paper, we extend the notion of a UI constraint to that of a *class-independent* (CI) constraint. In particular, every UI constraint is an instance of a CI constraint. Our second contribution is to demonstrate that patterns for UI constraints [5] can be generalized to patterns for CI constraints. The resulting algorithm, using these new patterns, remains FPT (irrespective of the authorization policy), although its running time is slower than that of the algorithm for WSP with UI constraints only. In short, our first two contributions identify a large class of constraints for which WSP is shown to be FPT, and subsume prior work in this area [9, 5, 14]. Our final contribution is an implementation of our algorithm in order to

investigate whether the theoretical advantages implied by its fixed-parameter tractability can be realized in practice. We compare our FPT algorithm with SAT4J, an off-the-shelf pseudo-Boolean (PB) SAT solver. The results of our experiments suggest that our FPT algorithm enjoys some significant advantages over SAT4J for hard instances of WSP.

In the next section, we define WSP and UI constraints in more formal terms, discuss related work in more detail, and introduce the notion of class-independent constraints. In Sections 3 and 4, we state and prove a number of technical results that underpin the algorithm for solving WSP with class-independent constraints. We describe the algorithm and establish its worst-case complexity in Section 5. In Section 6, we describe our experimental methods and report the results of our experiments. We conclude in Section 7.

Proofs of results marked with a $\star$ are given in Appendix A of the full version [8] of the paper. We also provide some details about the implementation of our algorithm in Appendix B of [8]. In the main body of the paper, we focus on the case of a single non-trivial partition of the user set. In Appendix C of [8], we generalize our approach to handle multiple (nested) partitions of the user set. (Such partitions can be used to model hierarchical organizational structures, which can be useful in practice [9].)

2 Workflow Satisfiability

Let $S = \{s_1, \ldots, s_k\}$ be a set of *steps*, let $U = \{u_1, \ldots, u_n\}$ be a set of *users* in a workflow specification, and let $k \leq n$. We are interested in assigning users to steps subject to certain constraints. In other words, among the set $\Pi(S, U)$ of functions from S to U, there are some that represent "legitimate" assignments of steps to users and some that do not.

The legitimacy or otherwise of an assignment is determined by the authorization policy and the constraints that complete the workflow specification. Let $\mathcal{A} = \{A(u) : u \in U\}$ be a set of *authorization lists*, where $A(u) \subseteq S$ for each $u \in U$, and let C be a set of *(workflow) constraints*. A *constraint* $c \in C$ may be viewed as a pair (T, Θ), where $T \subseteq S$ is the *scope* of c and Θ is a set of functions from T to U, specifying the assignments of steps in T to users in U that satisfy the constraint. In practice, we do not enumerate all the elements of Θ. Instead, we define its members implicitly using some constraint-specific syntax. In particular, we write (s, s', ρ), where $s, s' \in S$ and ρ is a binary relation defined on U, to denote a constraint that has scope $\{s, s'\}$ and is satisfied by any plan $\pi : S \to U$ such that $(\pi(s), \pi(s')) \in \rho$. Thus $(s, s', \neq)$, for example, requires s and s' to be performed by different users (and so represents a separation-of-duty constraint). Also $(s, s', =)$ states that s and s' must be performed by the same user (a binding-of-duty constraint).

2.1 The Workflow Satisfiability Problem

A *plan* is a function in $\Pi(S, U)$. Given a *workflow* $W = (S, U, \mathcal{A}, C)$, a plan π is *authorized* if for all $s \in S$, $s \in A(\pi(s))$, i.e. the user assigned to s is authorized for s. A plan π is *eligible* if for all $(T, \Theta) \in C$, $\pi|_T \in \Theta$, i.e. every constraint is satisfied. A plan π is *valid* if it is both authorized and eligible. In the *workflow satisfiability problem (WSP)*, we are given a workflow (specification) W, and our aim is to decide whether W has a valid plan. If W has a valid plan, W is *satisfiable*; otherwise, W is *unsatisfiable*.

Note that WSP is, in fact, the conservative CSP (i.e., CSP with unary constraints corresponding to step authorizations in the WSP terminology). However, unlike a typical instance of CSP, where the number of variables is significantly larger than the number of values, a typical instance of WSP has many more values (i.e., users) than variables (i.e., steps).

We assume that in all instances of WSP we consider, all constraints can be checked in time polynomial in n. Thus it takes polynomial time to check whether any plan is eligible. The correctness of our algorithm is unaffected by this assumption, but using constraints not checkable in polynomial time would naturally affect the running time.

▶ **Example 1.** Consider the following instance W' of WSP. The step and user sets are $S = \{s_1, s_2, s_3, s_4\}$ and $U = \{u_1, u_2, u_3, u_4, u_5\}$. The authorization lists are $A(u_1) = \{s_1, s_2, s_3, s_4\}$, $A(u_2) = \{s_1\}$, $A(u_3) = \{s_2\}$, $A(u_4) = A(u_5) = \{s_3, s_4\}$. The constraints are $(s_1, s_2, =)$, $(s_2, s_3, \neq)$, $(s_3, s_4, \neq)$, and $(s_4, s_1, \neq)$. Observe that $\pi' : S \to U$ with $\pi'(s_1) = \pi'(s_2) = u_1$, $\pi'(s_3) = u_5$ and $\pi'(s_4) = u_4$ satisfies all constraints and authorizations, and thus π' is a valid plan for W'. Therefore, W' is satisfiable.

2.2 Constraints using Equivalence Relations

Crampton *et al.* [9] introduced constraints defined in terms of an equivalence relation $\sim$ on U: a plan π satisfies constraint $(s, s', \sim)$ if $\pi(s) \sim \pi(s')$ (and satisfies constraint $(s, s', \nsim)$ if $\pi(s) \nsim \pi(s')$). Hence, we could, for example, specify the pair of constraints $(s, s', \neq)$ and $(s, s', \sim)$, which, collectively, require that s and s' are performed by different users that belong to the same equivalence class. As we noted in the introduction, such constraints are very natural in the context of organizations that partition the set of users into departments, groups or teams.

Moreover, Crampton *et al.* [9] demonstrated that "nested" equivalence relations can be used to model hierarchical structures within an organization[1] and to define constraints on workflow execution with respect to those structures. More formally, an equivalence relation $\sim$ is said to be a *refinement* of an equivalence relation $\approx$ if $x \sim y$ implies $x \approx y$. In particular, given an equivalence relation $\sim$, $=$ is a refinement of $\sim$. Crampton *et al.* proved that WSP remains FPT when some simple extensions of constraints $(s, s', \sim)$ and $(s, s', \nsim)$ are included [9, Theorem 5.4]. Our extension of constraints $(s, s', \sim)$ and $(s, s', \nsim)$ is much more general: it is similar to generalizing simple constraints $(s, s', =)$ and $(s, s', \neq)$ to the wide class of UI constraints. This leads, in particular, to a significant generalization of Theorem 5.4 in [9].

Let $c = (T, \Theta)$ be a constraint and let $\sim$ be an equivalence relation on U. Let $U^{\sim}$ denote the set of equivalence classes induced by $\sim$ and let $u^{\sim} \in U^{\sim}$ denote the equivalence class containing u. Then, for any function $\pi : S \to U$, we may define the function $\pi^{\sim} : S \to U^{\sim}$, where $\pi^{\sim}(s) = (\pi(s))^{\sim}$. In particular, $\sim$ induces a set of functions $\Theta^{\sim} = \{\theta^{\sim} : \theta \in \Theta\}$.

▶ **Example 2.** Continuing from Example 1, suppose $U^{\sim}$ consists of two equivalence classes $U_1 = \{u_1, u_2, u_5\}$ and $U_2 = \{u_3, u_4\}$. Let us add to W' another constraint $(s_1, s_4, \sim)$ (s_1 and s_4 must be assigned users from the same equivalence class) to form a new instance W'' of WSP. Then plan π' does not satisfy the added constraint and so π' is not valid for W''. However, $\pi'' : S \to U$ with $\pi''(s_1) = \pi''(s_2) = u_1$, $\pi''(s_3) = u_4$ and $\pi''(s_4) = u_5$ satisfies all constraints and authorizations, and thus π'' is valid for W''. Here $(\pi'')^{\sim}(s_1) = (\pi'')^{\sim}(s_2) = (\pi'')^{\sim}(s_4) = U_1$ and $(\pi'')^{\sim}(s_3) = U_2$.

Given an equivalence relation $\sim$ on U, we say that a constraint $c = (T, \Theta)$ is *class-independent (CI) for* $\sim$ if $\theta^{\sim} \in \Theta^{\sim}$ implies $\theta \in \Theta$, and for any permutation $\phi : U^{\sim} \to U^{\sim}$, $\theta^{\sim} \in \Theta^{\sim}$ implies $\phi \circ \theta^{\sim} \in \Theta^{\sim}$. In other words, if a plan $\pi : s \mapsto \pi(s)$ satisfies a constraint c,

[1] Many organizations exhibit nested hierarchical structure. For example, the academic parts of many universities are divided into faculties/schools which are divided into departments.

which is class-independent for $\sim$, then for each permutation ϕ of classes in $U^{\sim}$ if we replace $\pi(s)$ by any user in the class $\phi(\pi(s)^{\sim})$ for every step s, then the new plan will satisfy c.

We say a constraint is *user-independent (UI)* if it is CI for $=$. In other words, if a plan $\pi : s \mapsto \pi(s)$ satisfies a UI constraint c and we replace any user in $\{\pi(s) : s \in S\}$ by an arbitrary user such that the replacement users are all distinct, then the new plan satisfies c.

We conclude this section with a claim whose simple proof is omitted.

▶ **Proposition 3.** *Given two equivalence relations $\sim$ and $\approx$ such that $\sim$ is a refinement of $\approx$, and any plan $\pi : S \to U$, $\pi^{\sim}(s) = \pi^{\sim}(s')$ implies $\pi^{\approx}(s) = \pi^{\approx}(s')$.*

3 Plans and Patterns

In what follows, unless specified otherwise, we will consider the equivalence relation $=$ along with another fixed equivalence relation $\sim$. We will write $[m]$ to denote the set $\{1, \ldots, m\}$ for any integer $m \geqslant 1$. We assume that all constraints are either UI or CI for $\sim$. For brevity, we will refer to constraints that are CI for $\sim$ as simply *CI*. We consider only two equivalence relations for simplicity of presentation, but our results below can be generalized to any sequence $\sim_1, \ldots, \sim_l$ of equivalence relations such that $\sim_{i+1}$ is a refinement of $\sim_i$ for all $i \in [l-1]$, see Appendix C in the full version [8] of the paper. It is important to keep in mind that we put *no restrictions on authorizations*.

We will represent groups of plans as *patterns*. The intuition is that a pattern defines a partition of the set of steps relevant to a set of constraints. For instance, suppose that we only have UI constraints. Then a pattern specifies which sets of steps are to be assigned to the same user. A pattern assigns an integer to each step and those steps that are labelled by the same integer will be mapped to the same user. A pattern p defines an equivalence relation $\sim_p$ on the set of steps (where $s \sim_p s'$ if and only if s and s' are assigned the same label). Moreover, this pattern can be used to define a plan by mapping each of the equivalence classes induced by $\sim_p$ to a different user. Since we only consider UI constraints, the identities of the users are irrelevant (provided they are distinct). Conversely, any plan $\pi : S \to U$ defines a pattern: s and s' are labelled with the same integer if and only if $\pi(s) = \pi(s')$. And if π satisfies a UI constraint c, then any other plan with the same pattern will also satisfy c. We can extend this notion of a pattern to CI constraints where entries in the pattern encode equivalence classes of users instead of single users.

More formally, let $W = (S, U, \mathcal{A}, C = C_{=} \cup C_{\sim})$ be a workflow, where $C_{=}$ is a set of UI constraints and $C_{\sim}$ is a set of CI constraints. Let $p_{=} = (x_1, \ldots, x_k)$ where $x_i \in [k]$ for all $i \in [k]$. We say that $p_{=}$ is a *UI-pattern* for a plan π if $x_i = x_j \Leftrightarrow \pi(s_i) = \pi(s_j)$, for all $i, j \in [k]$, and $p_{=}$ is *eligible for* $C_{=}$ if any plan π with $p_{=}$ as its UI-pattern is eligible for $C_{=}$.

In Example 2, $C_{=} = \{(s_1, s_2, =), (s_2, s_3, \neq), (s_3, s_4, \neq), (s_1, s_4, \neq)\}$ and $C_{\sim} = \{(s_1, s_4, \sim)\}$. Tuples $(1, 1, 2, 3)$ and $(2, 2, 4, 3)$ are UI-patterns for plan π'' of Example 2.

▶ **Proposition 4** (⋆). *Let $p_{=}$ be a UI-pattern for a plan π. Then $p_{=}$ is eligible for $C_{=}$ if and only if π is eligible for $C_{=}$.*

Let $p_{\sim} = (y_1, \ldots, y_k)$, where $y_i \in [k]$ for all $i \in [k]$. We say that $p_{\sim}$ is a *CI-pattern* for a plan π if $y_i = y_j \Leftrightarrow \pi^{\sim}(s_i) = \pi^{\sim}(s_j)$, for all $i, j \in [k]$, and $p_{\sim}$ is *eligible for* $C_{\sim}$ if any plan π with $p_{\sim}$ as its CI-pattern is eligible for $C_{\sim}$. For example, $(1, 1, 2, 1)$ and $(2, 2, 4, 2)$ are CI-patterns for plan π'' of Example 2. The next result is a generalization of Proposition 4.

▶ **Proposition 5** (⋆). *Let $p_{\sim}$ be a CI-pattern for a plan π. Then $p_{\sim}$ is eligible for $C_{\sim}$ if and only if π is eligible for $C_{\sim}$.*

Now let $p = (p_=, p_\sim)$ be a pair containing a UI-pattern and an CI-pattern. Then we call p a *(UI, CI)-pattern*. We say that p is a (UI, CI)-pattern for π if $p_=$ is a UI-pattern for π and $p_\sim$ is a CI-pattern for π. We say that p is *eligible for* $C = C_= \cup C_\sim$ if $p_=$ is eligible for $C_=$ and $p_\sim$ is eligible for $C_\sim$. The following two results follow immediately from Propositions 4 and 5 and definitions of UI- and CI-patterns.

▶ **Lemma 6.** *Let $p = (p_=, p_\sim)$ be a (UI, CI)-pattern for a plan π. Then p is eligible for $C = C_= \cup C_\sim$ if and only if π is eligible for C.*

▶ **Proposition 7.** *There is a (UI, CI)-pattern p for every plan π.*

We say a (UI, CI)-pattern p is *realizable* if there exists a plan π such that π is authorized and p is a (UI, CI)-pattern for π. Given the above results, in order to solve a WSP instance with user- and class-independent constraints, it is enough to decide whether there exists a (UI, CI)-pattern p such that (i) p is realizable, and (ii) p is eligible (and hence π is eligible) for $C = C_= \cup C_\sim$.

We will enumerate all possible (UI, CI)-patterns, and for each one check whether the two conditions hold. We defer the explanation of how to determine whether p is realizable until Sec. 4. We now show it is possible to check whether a (UI, CI)-pattern $p = (p_=, p_\sim)$ is eligible in time polynomial in the input size N. Indeed, in polynomial time, we can construct plans $\pi_=$ and $\pi_\sim$ with patterns $p_=$ and $p_\sim$, respectively, where $\pi_=(s_i) = \pi_=(s_j)$ if and only if $x_i = x_j$ and $\pi_\sim(s_i) \sim \pi_\sim(s_j)$ if and only if $y_i = y_j$. (In particular, we can select a representative user from each equivalence class in $U^\sim$.) By Lemma 6 and Propositions 4 and 5, p is eligible if and only if both $\pi_=$ and $\pi_\sim$ are eligible. By our assumption before Example 1, eligibility of both $\pi_=$ and $\pi_\sim$ can be checked in polynomial time.[2] Note, however, that $\pi_=$ and $\pi_\sim$ may be different plans, so this simple check for eligibility does not give us a check for realizability of p.

4 Checking Realizability

A *partial plan* π is a function from a subset T of S to U. In particular, a plan is a partial plan. To avoid confusion with partial plans, sometimes we will call plans *complete plans*. We can easily extend the definitions of *eligible, authorized* and *valid* plans to partial plans: the only difference is that we only consider authorizations for steps in T and constraints with scope being a subset of T.

We also define *partial patterns*. For a UI or CI-pattern $q = (x_1, \ldots, x_k)$ and a subset $T \subseteq S$, let the pattern $q|_T = (z_1, \ldots, z_k)$, where $z_i = x_i$ if $s_i \in T$, and $z_i = 0$ otherwise. We say that $p|_T$ is a (UI, CI)-pattern for a partial plan $\pi : T \to U$ if $p|_T$ with all coordinates with 0 values removed is a (UI, CI)-pattern for π. We therefore have that if p is a (UI, CI)-pattern for a plan π, then $p|_T$ is a (UI, CI)-pattern for π restricted to T.

Let $p = (p_= = (x_1, \ldots, x_k), p_\sim = (y_1, \ldots, y_k))$ be a (UI, CI)-pattern. We say that p is *consistent* if $x_i = x_j \Rightarrow y_i = y_j$ for all $i, j \in [k]$. Recall that if p is the (UI, CI)-pattern for π, then $x_i = x_j \Leftrightarrow \pi(s_i) = \pi(s_j)$, and $y_i = y_j \Leftrightarrow \pi^\sim(s_i) = \pi^\sim(s_j)$. Thus Proposition 3 implies that if p is the (UI, CI)-pattern for any plan then p is consistent. Henceforth, we will only consider (UI, CI)-patterns that are consistent.

Given a (UI, CI)-pattern $(p_=, p_\sim)$, we must determine whether this (UI, CI)-pattern can be realized, given the authorization lists defined on users. The patterns $p_=$ and $p_\sim$ define

[2] Clearly, it is not hard to check eligibility of p without explicitly constructing $\pi_=$ and $\pi_\sim$, as is done in our algorithm implementation, described in Appendix B of [8].

two sets of equivalence classes on S: s_i and s_j are in the same equivalence class of S defined by $p_=$ ($p_\sim$, respectively) if and only if $x_i = x_j$ ($y_i = y_j$, respectively).

Moreover each equivalence class induced by $p_\sim$ is partitioned by equivalence classes induced by $p_=$. We must determine whether there exists a plan $\pi : S \to U$ that simultaneously (i) has UI-pattern $p_=$; (ii) has CI-pattern $p_\sim$; and (iii) assigns an authorized user to each step. Informally, our algorithm for checking realizability computes two things.

- For each pair (T, V), where $T \subseteq S$ is an equivalence class induced by $p_\sim$ and $V \subseteq U$ is an equivalence class induced by $\sim$, whether there exists an injective mapping from the equivalence classes in T induced by $p_=$ to authorized users in V. We call such a mapping a *second-level* mapping.
- Whether there exists an injective mapping f from the set of equivalence classes induced by $p_\sim$ to the set of equivalence classes induced by $\sim$ such that $f(T) = V$ if and only if there exists a second-level mapping from T to V. We call f a *top-level* mapping.

If a top-level mapping exists, then, by construction, it can be "deconstructed" into authorized partial plans defined by second-level mappings. We compute top- and second-level mappings using matchings in bipartite graphs, as described below.

The Top-level Bipartite Graph. The UI-pattern $p_= = (x_1, \ldots, x_k)$ induces an equivalence relation on $S = \{s_1, \ldots, s_k\}$, where s_i and s_j are equivalent if and only if $x_i = x_k$. Let $\mathcal{S} = \{S_1, \ldots, S_l\}$ be the set of equivalence classes of S under this relation. Similarly, the CI-pattern $p_\sim = (y_1, \ldots, y_k)$ induces an equivalence relation on S, where s_i, s_j are equivalent if and only if $y_i = y_j$. Let $\mathcal{T} = \{T_1, \ldots, T_m\}$ be the equivalence classes under this relation. Observe that since p is consistent, we have $k \geq l \geq m$ and for any S_i, T_j, either $S_i \subseteq T_j$ or $S_i \cap T_j = \emptyset$.

▶ **Definition 8.** Given a (UI, CI)-pattern $p = (p_=, p_\sim)$, the *top-level bipartite graph* G_p is defined as follows. Let the partite sets of G_p be $\mathcal{T}$ and $U^\sim$. For each $T_r \in \mathcal{T}$ and class $u^\sim$, we have an edge between T_r and $u^\sim$ if and only if there exists an authorized partial plan $\pi_r : T_r \to u^\sim$ such that $p_=|_{T_r}$ is a UI-pattern for π_r.

▶ **Lemma 9.** *If a (UI, CI)-pattern $p = (p_=, p_\sim)$ is realizable, then G_p has a matching covering $\mathcal{T}$.*

Proof. Let π be an authorized plan such that p is a (UI, CI)-pattern for π. As $p_\sim$ is a CI-pattern for π, we have that for each $T_r \in \mathcal{T}$ and all $s_i, s_j \in T_r$, $\pi^\sim(s_i) = \pi^\sim(s_j)$. Therefore $\pi(T_r) \subseteq u^\sim$ for some $u \in U$. Let $u_r^\sim$ be this equivalence class for each T_r. As $p_\sim$ is a CI-pattern for π, we have that for all $r \neq r'$ and any $s_i \in T_r, s_j \in T_{r'}$, $\pi^\sim(s_i) \neq \pi^\sim(s_j)$. It follows that $u_r^\sim \neq u_{r'}^\sim$ for any $r \neq r'$.

Let $M = \{T_r u_r^\sim \in E(G_p) : T_r \in \mathcal{T}\}$. As $u_r^\sim \neq u_{r'}^\sim$ for any $r \neq r'$ we have that M is a matching that covers $\mathcal{T}$. It remains to show that M is a matching of G_p covering $\mathcal{T}$, i.e. that $T_r u_r^\sim$ is an edge in G_p for each T_r. For each $T_r \in \mathcal{T}$, let π_r be π restricted to T_r. Then π_r is a function from T_r to $u_r^\sim$. As π is authorized, π_r is also authorized. As $p_=$ is a UI-pattern for π, we have that $p_=|_{T_r}$ is a UI-pattern for π_r. Therefore π_r satisfies all the conditions for there to be an edge $T_r u_r^\sim$ in G_p. ◀

We have shown that for any (UI, CI)-pattern to be realizable, it must be consistent and its top-level bipartite graph must have a matching covering $\mathcal{T}$. We will now show that these necessary conditions are also sufficient.

▶ **Lemma 10 (⋆).** *Let $p = (p_= = (x_1, \ldots, x_k), p_\sim = (y_1, \ldots, y_k))$ be a (UI, CI)-pattern which is consistent, and such that G_p has a matching covering $\mathcal{T}$. Then p is realizable.*

The Second-level Bipartite Graph. For each (UI, CI)-pattern $p = (p_=, p_\sim)$, we need to construct the graph G_p and decide whether it has a matching covering $\mathcal{T}$, in order to decide whether p is realizable. Given G_p, a maximum matching can be found in polynomial time using standard techniques, but constructing G_p itself is non-trivial. For each potential edge $T_r u^\sim$ in G_p, we need to decide whether there exists an authorized partial plan $\pi_r : T_r \to u^\sim$ such that $p_=|_{T_r}$ is a UI-pattern for π_r. We can decide this by constructing another bipartite graph, $G_{T_r u^\sim}$. Recall that $\mathcal{S} = \{S_1, \ldots, S_l\}$ is a partition of S into equivalence classes, where s_i, s_j are equivalent if $x_i = x_j$, and for each $S_h \in \mathcal{S}$, either $S_h \subseteq T_r$ or $S_h \cap T_r = \emptyset$. Define $\mathcal{S}_r = \{S_h : S_h \subseteq T_r\}$.

▶ **Definition 11.** Given a (UI, CI)-pattern $p = (p_= = (x_1, \ldots, x_k), p_\sim = (y_1, \ldots, y_k))$, a set $T_r \in \mathcal{T}$ and equivalence class $u^\sim \in U^\sim$, the *second-level bipartite graph* $G_{T_r u^\sim}$ is defined as follows: Let the partite sets of G be $\mathcal{S}_r$ and $u^\sim$ and for each $S_h \in \mathcal{S}_r$ and $v \in u^\sim$, we have an edge between S_h and v if and only if v is authorized for all steps in S_h.

▶ **Lemma 12 (⋆).** *Given $T_r \in \mathcal{T}$, $u^\sim \in U^\sim$, the following conditions are equivalent.*

- *There exists an authorized partial plan $\pi : T_r \to u^\sim$ such that $p_=|_{T_r}$ is a UI-pattern for π.*
- *$G_{T_r u^\sim}$ has a matching that covers $\mathcal{S}_r$.*

5 FPT Algorithm

Our FPT algorithm generates (UI, CI)-patterns p in a backtracking manner as follows. (Its implementation is described in Appendix B of [8].) It first generates partial patterns $p_= = (x_1, \ldots, x_k)$, where the coordinates $x_i = 0$ are assigned one by one to integers in $[k']$, where $k' = \max_{1 \le j \le k}\{x_j\} + 1$. The algorithm checks that the pattern $p_=$ does not violate any constraints whose scope contain the corresponding step s_i. If an eligible pattern $p_= = (x_1, \ldots, x_k)$ has been completed (i.e., $x_j \neq 0$ for each $j \in [k]$), the partial patterns $p_\sim = (y_1, \ldots, y_k)$ are generated as above but with a difference: the algorithm ensures the consistency condition. If an eligible (UI, CI)-pattern p has been constructed, a procedure constructing bipartite graphs and searching for matchings in them as described in Section 4 decides whether p is realizable.

▶ **Theorem 13.** *We can solve WSP with UI and CI constraints in $O^*(4^{k \log_2 k})$ time.*

Proof Sketch. Our algorithm is correct by Proposition 7 and the fact that every complete (UI, CI)-pattern can be generated. It remains to estimate the running time.

If $p_\sim$ in our algorithm were generated as $p_=$, i.e., consistency were not taken into consideration, the search tree $\mathcal{T}$ of our algorithm (nodes are partial (UI, CI)-patterns) would have at least as many nodes as the actual search tree of our algorithm. Observe that each internal node (corresponding to an incomplete (UI, CI)-pattern) in $\mathcal{T}$ has at least two children, and each leaf in this tree corresponds to a complete (UI, CI)-pattern. Thus, the total number of partial (UI, CI)-patterns considered by our algorithm is less than twice the number of complete (UI, CI)-patterns, which is $k^{2k} = 4^{k \log k}$ as each of $2k$ coordinates takes values in $[k]$.

We have to compute a matching in the top-level bipartite graph and matchings for each second-level bipartite graph. The number of second-level bipartite graphs is bounded above by nk (since the number of equivalence classes in U and S cannot exceed n and k, respectively). We can compute a maximum matching in time polynomial in the number of vertices in the top- and second-level bipartite graphs (which is bounded in all our graphs by $n + k$). The result follows. ◀

6 Algorithm Implementation and Computational Experiments

There can be a huge difference between an algorithm in principle and its actual implementation as a computer code, e.g., see [13]. We have implemented the new pattern-backtracking FPT algorithm and a reduction to the pseudo-Boolean satisfiability (PB SAT) problem in C++, using SAT4J [12] as a pseudo-Boolean SAT solver. Reductions from WSP constraints to PB ones were done similarly to those in [4, 6, 11]. Our FPT algorithm extends the pattern-backtracking framework of [11] in a nontrivial way, for details see Appendix B of [8].

In this section we describe a series of experiments that we ran to test the performance of our FPT algorithm against that of SAT4J. Due to the difficulty of acquiring real-world workflow instances, we generate and use synthetic data to test our new FPT algorithm and reduction to the PB SAT problem (as in similar experimental studies [6, 11, 14]). All our experiments use a MacBook Pro computer having a 2.6 GHz Intel Core i5 processor, 8 GB 1600 MHz DDR3 RAM and running Mac OS X 10.9.5.

We generate a number of random WSP instances using not-equals (i.e, constraints of the form $(s, s', \neq)$), equivalence and non-equivalence constraints (i.e., constraints of the types $(s, s', \sim)$ and $(s, s', \nsim)$), and at-most constraints. An *at-most constraint* is a UI constraint that restricts the number of users that may be involved in the execution of a set of steps. It is, therefore, a form of cardinality constraint and imposes a loose form of "need-to-know" constraint on the execution of a workflow instance, which can be important in certain business processes. An at-most constraint may be represented as a tuple $(t, Q, \leqslant)$, where $Q \subseteq S$, $1 \leqslant t \leqslant |Q|$, and is satisfied by any plan that allocates no more than t users in total to the steps in Q. In all our at-most constraints $t = 3$ and $|Q| = 5$ as in [6, 11].

6.1 Experimental Parameters and Instance Generation

We summarize the parameters we use for our experiments in Table 1. Values of k, n and r were chosen that seemed appropriate for real-world workflow specifications. The values of the other parameters were determined by preliminary experiments designed to identify "challenging" instances of WSP: that is, instances that were neither very lightly constrained nor very tightly constrained. Informally, it is relatively easy to determine that lightly constrained instances are satisfiable and that tighly constrained instances are unsatisfiable. Thus the instances we use in our experiments are (very approximately) equally likely to be satisfiable or unsatisfiable. In particular, by varying the numbers of at-most constraints and constraints of the form $(s, s', \nsim)$, we are able to generate a set of instances with the desired characteristics (as shown by the results in Table 2).

A constraint $(s, s', \nsim)$ implies the existence of a constraint $(s, s', \neq)$, so we do not vary the number of not-equals a great deal (in contrast to existing work in the literature [6]). Informally, a constraint $(s, s', \sim)$ reduces the difficulty of finding a valid plan. Thus, given our desire to investigate challenging instances, we do not use very many of these constraints.

All the constraints, authorizations, and equivalence classes of users are generated for each instance separately, uniformly at random. The random generation of authorizations, not-equals, and at-most constraints uses existing techniques [6]. The generation of equivalence and non-equivalence constraints has to be controlled to ensure that an instance is not trivially unsatisfiable. In particular, we must discard a constraint of the form $(s, s', \nsim)$ if we have already generated a constraint of the form $(s, s', \sim)$. The equivalence classes of the user set are generated by enumerating the user set and then splitting the list into contiguous sublists. The number of elements in each sublist varies between 3 and 7 (chosen uniformly at random and adjusted, where necessary, so that the total number of members in the r sub-lists is n).

Table 1 Parameters used in our experiments.

Parameter		Values
Number of steps k		$20, 25, 30$
Number of users n		$10k$
Number of user equivalence classes r		$2k$
Number of constraints $(s, s', \neq)$	$k = 20$	$20, 25$
	$k = 25$	$25, 30$
	$k = 30$	$30, 35$
Number of constraints $(s, s', \sim)$	$k = 20$	0
	$k = 25$	1
	$k = 30$	2
Number of constraints $(s, s', \nsim)$	$k = 20$	$10, 15, 20, 25, 30$
	$k = 25$	$15, 20, 25, 30, 35$
	$k = 30$	$20, 25, 30, 35, 40$
Number of at-most constraints	$k = 20$	$10, 15, 20, 25, 30, 35, 40$
	$k = 25$	$15, 20, 25, 30, 35, 40, 45$
	$k = 30$	$20, 25, 30, 35, 40, 45, 50$

6.2 Results and Evaluation

We adopt the following labelling convention for our test instances: $a.b.c.d$ denotes an instance with a not-equals constraints, b at-most constraints, c equivalence constraints, and d non-equivalence constraints (as used in the first and fourth columns of Table 2, for instances with $k = 25$ and $k = 30$, respectively). In our experiments we compare the run-times and outcomes of SAT4J (having reduced the WSP instance to a PB SAT problem instance) and our FPT algorithm, which we will call PBA4CI (pattern-based algorithm for class-independent constraints). Table 2 shows some detailed results of our experiments (the results for $k = 20$ were excluded due to the space limit). We record whether an instance is solved, indicating a satisfiable instance with a 'Y' and an unsatisfiable instance with a 'N'; instances that were not solved are indicated by a question mark. PBA4CI reaches a conclusive decision (Y or N) for every test instance, whereas SAT4J fails to reach such a decision for some instances, typically because the machine runs out of memory. The table also records the time (in seconds) taken for the algorithms to run on each instance. We would expect that the time taken to solve an instance would depend on whether the instance is satisfiable or not, and this is confirmed by the results in the table.

In total, the experiments cover 210 randomly generated instances, 70 instances for each number of steps, $k \in \{20, 25, 30\}$. PBA4CI successfully solves all of the instances, while SAT4J fails on almost 40% of the instances (mostly unsatisfiable ones). In terms of CPU time, SAT4J is more efficient only on 5 instances (2.4%) in total: 1 for 20 steps, 1 for 25 steps, and 3 for 30 steps, all of which are lightly constrained. For these instances PBA4CI has to generate a large number of patterns in the search space before it finds a solution.

Overall, PBA4CI is clearly more effective and efficient than SAT4J on these instances. Table 3 put in Appendix B of [8] shows the summary statistics for all the experiments. The numbers of unsolved instances by SAT4J are indicated in parenthesis. For average CPU time values, we assume that the running time on the unsolved instances can be considered as a lower bound on the time required to solve them. Therefore average time values in Table 3 take into consideration unsolved instances for SAT4J: they are estimated lower bounds on its

Table 2 Results for $k = 25$ and 30. Time in seconds. Y,N,? mean satisfied, unsatisfied, unsolved.

Instance	SAT4J		PBA4CI		Instance	SAT4J		PBA4CI	
	$k = 25$					$k = 30$			
25.15.1.15	Y	2.62	Y	2.464	30.20.2.20	Y	2.72	Y	50.804
25.20.1.15	Y	22.38	Y	0.010	30.25.2.20	Y	271.78	Y	2.323
25.25.1.15	Y	11.03	Y	0.010	30.30.2.20	?	2,141.60	Y	2.946
25.30.1.15	Y	35.54	Y	0.040	30.35.2.20	?	2,250.02	N	0.412
25.35.1.15	N	1,439.94	N	0.075	30.40.2.20	?	1,942.57	N	2.238
25.40.1.15	?	2,088.06	N	0.033	30.45.2.20	?	2,198.02	N	2.171
25.45.1.15	Y	113.37	Y	0.022	30.50.2.20	?	2,580.81	N	0.494
25.15.1.20	Y	1.52	Y	111.799	30.20.2.25	Y	4.18	Y	237.604
25.20.1.20	Y	7.77	Y	0.024	30.25.2.25	Y	76.41	Y	0.789
25.25.1.20	Y	297.39	Y	0.065	30.30.2.25	?	2,288.07	N	0.401
25.30.1.20	?	2,273.56	N	0.033	30.35.2.25	Y	1,364.66	Y	0.238
25.35.1.20	Y	48.29	Y	0.067	30.40.2.25	?	2,383.92	N	0.775
25.40.1.20	N	105.48	N	0.045	30.45.2.25	?	1,743.87	N	0.394
25.45.1.20	?	2,105.61	N	0.031	30.50.2.25	?	2,385.39	N	0.218
25.15.1.25	Y	14.40	Y	0.014	30.20.2.30	Y	35.40	Y	0.071
25.20.1.25	Y	80.25	Y	0.021	30.25.2.30	Y	9.37	Y	1.063
25.25.1.25	?	2,284.78	N	0.023	30.30.2.30	N	1,632.51	N	0.347
25.30.1.25	N	442.91	N	0.237	30.35.2.30	Y	803.50	Y	0.029
25.35.1.25	?	2,188.01	N	0.060	30.40.2.30	?	2,022.71	N	0.981
25.40.1.25	?	2,293.77	N	0.043	30.45.2.30	?	1,902.84	N	1.501
25.45.1.25	?	2,041.02	N	0.144	30.50.2.30	?	1,730.93	N	0.467
25.15.1.30	Y	3.22	Y	0.011	30.20.2.35	Y	24.12	Y	0.453
25.20.1.30	Y	240.59	Y	0.014	30.25.2.35	Y	456.51	Y	0.085
25.25.1.30	Y	66.74	Y	0.050	30.30.2.35	N	1,817.76	N	1.088
25.30.1.30	?	2,301.75	N	0.088	30.35.2.35	?	1,949.77	N	0.111
25.35.1.30	N	1,562.30	N	0.023	30.40.2.35	?	2,115.32	N	0.551
25.40.1.30	?	2,332.07	N	0.127	30.45.2.35	?	1,535.57	N	0.118
25.45.1.30	N	950.25	N	0.040	30.50.2.35	?	1,647.41	N	0.454
25.15.1.35	Y	10.57	Y	0.014	30.20.2.40	?	3,088.54	N	0.729
25.20.1.35	N	218.70	N	0.166	30.25.2.40	?	1,746.81	Y	0.542
25.25.1.35	Y	37.87	Y	0.012	30.30.2.40	?	2,350.01	Y	0.949
25.30.1.35	?	2,421.30	N	0.054	30.35.2.40	?	1,857.27	N	0.576
25.35.1.35	N	1,524.68	N	0.022	30.40.2.40	?	1,938.63	N	0.221
25.40.1.35	N	1,001.67	N	0.028	30.45.2.40	?	2,159.50	N	0.209
25.45.1.35	?	1,974.05	N	0.034	30.50.2.40	?	1,815.15	N	0.337

average time performance. As the number of steps k increases, SAT4J fails more frequently and is unable to reach a conclusive decision for more than 65% of instances when $k = 30$, some of which are satisfiable. However, SAT4J is clearly more efficient (and effective) on satisfiable instances than on the unsatisfiable ones, while for PBA4CI the converse is true. This can be explained by very different search strategies used by the solvers.

7 Conclusion

We have introduced the concept of a class-independent constraint, which significantly generalizes user-independent constraints and substantially extends the range of real-world business requirements that can be modelled. We have designed an FPT algorithm for WSP with

class-independent constraints. Our computational results demonstrate that our FPT algorithm is useful in practice for WSP with class-independent constraints, in particular for WSP instances that are too hard for SAT4J.

The full generalization of our approach is briefly described in Appendix C of [8] and the time complexity of the corresponding algorithm, $O^*(2^{rk \log_2 k})$ (r is the number of nested equivalence relations including $=$), indicates that it will remain practical at least when three rather than two equivalence relations are considered.

Acknowledgement. This research was supported by an EPSRC grant EP/K005162/1. The FPT algorithm's executable code and experimental data set are publicly available [10].

References

1 American National Standards Institute. *ANSI INCITS 359-2004 for Role Based Access Control*, 2004.
2 D. A. Basin, S. J. Burri, and G. Karjoth. Obstruction-free authorization enforcement: Aligning security and business objectives. *J. Comput. Security*, 22(5):661–698, 2014.
3 D. F. C. Brewer and M. J. Nash. The Chinese Wall security policy. In *IEEE Symposium on Security and Privacy*, pages 206–214. IEEE Computer Society, 1989.
4 D. Cohen, J. Crampton, A. Gagarin, G. Gutin, and M. Jones. Engineering algorithms for workflow satisfiability problem with user-independent constraints. In J. Chen, J.E. Hopcroft, and J. Wang, editors, *Frontiers in Algorithmics, FAW 2014*, volume 8497 of *Lecture Notes in Computer Science*, pages 48–59. Springer, 2014.
5 D. Cohen, J. Crampton, A. Gagarin, G. Gutin, and M. Jones. Iterative plan construction for the workflow satisfiability problem. *J. Artif. Intel. Res.*, 51:555–577, 2014.
6 D. Cohen, J. Crampton, A. Gagarin, G. Gutin, and M. Jones. Algorithms for the workflow satisfiability problem engineered for counting constraints. *J. Comb. Optim.*, to appear, 2015. (DOI: 10.1007/s10878-015-9877-7).
7 J. Crampton. A reference monitor for workflow systems with constrained task execution. In E. Ferrari and G.-J. Ahn, editors, *SACMAT*, pages 38–47. ACM, 2005.
8 J. Crampton, A. V. Gagarin, G. Gutin, and M. Jones. On the workflow satisfiability problem with class-independent constraints. *CoRR*, abs/1504.03561, 2015.
9 J. Crampton, G. Gutin, and A. Yeo. On the parameterized complexity and kernelization of the workflow satisfiability problem. *ACM Trans. Inf. Syst. Secur.*, 16(1):4, 2013.
10 A. Gagarin, J. Crampton, G. Gutin, and M. Jones. Implementation of the pattern-backtracking FPT algorithm and experimental data set for the WSP with class-independent constraints. `http://dx.doi.org/10.6084/m9.figshare.1502692`, Aug 2015.
11 D. Karapetyan, A. Gagarin, and G. Gutin. Pattern backtracking algorithm for the workflow satisfiability problem. In *Frontiers in Algorithmics 2015*, volume 9130 of *Lect. Notes Comput. Sci.*, pages 138–149. Springer, 2015.
12 D. Le Berre and A. Parrain. The SAT4J library, release 2.2. *J. Satisf. Bool. Model. Comput.*, 7:59–64, 2010.
13 W. Myrvold and W. Kocay. Errors in graph embedding algorithms. *J. Comput. Syst. Sci.*, 77(2):430–438, 2011.
14 Q. Wang and N. Li. Satisfiability and resiliency in workflow authorization systems. *ACM Trans. Inf. Syst. Secur.*, 13(4):40, 2010.

Parameterized Algorithms for Min-Max Multiway Cut and List Digraph Homomorphism*†

Eunjung Kim[1], Christophe Paul[2], Ignasi Sau[2], and Dimitrios M. Thilikos[2,3,4]

1 **CNRS, LAMSADE, Paris, France**
eunjungkim78@gmail.com
2 **AlGCo project-team, CNRS, LIRMM, Montpellier, France**
{Christophe.Paul,Ignasi.Sau}@lirmm.fr, sedthilk@thilikos.info
3 **Department of Mathematics, University of Athens, Athens, Greece**
4 **Computer Technology Institute & Press "Diophantus", Patras, Greece**

Abstract

In this paper we design FPT-algorithms for two parameterized problems. The first is List Digraph Homomorphism: given two digraphs G and H and a list of allowed vertices of H for every vertex of G, the question is whether there exists a homomorphism from G to H respecting the list constraints. The second problem is a variant of Multiway Cut, namely Min-Max Multiway Cut: given a graph G, a non-negative integer ℓ, and a set T of r terminals, the question is whether we can partition the vertices of G into r parts such that (a) each part contains one terminal and (b) there are at most ℓ edges with only one endpoint in this part. We parameterize List Digraph Homomorphism by the number w of edges of G that are mapped to non-loop edges of H and we give a time $2^{O(\ell \cdot \log h + \ell^2 \cdot \log \ell)} \cdot n^4 \cdot \log n$ algorithm, where h is the order of the host graph H. We also prove that Min-Max Multiway Cut can be solved in time $2^{O((\ell r)^2 \log \ell r)} \cdot n^4 \cdot \log n$. Our approach introduces a general problem, called List Allocation, whose expressive power permits the design of parameterized reductions of both aforementioned problems to it. Then our results are based on an FPT-algorithm for the List Allocation problem that is designed using a suitable adaptation of the *randomized contractions* technique (introduced by [Chitnis, Cygan, Hajiaghayi, Pilipczuk, and Pilipczuk, FOCS 2012]).

1998 ACM Subject Classification G.2.1 Combinatorics, G.2.2 Graph Theory

Keywords and phrases Parameterized complexity, Fixed-Parameter Tractable algorithm, Multiway Cut, Digraph homomorphism

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.78

1 Introduction

The Multiway Cut problem asks, given a graph G, a set of r terminals T, and a non-negative integer ℓ, whether it is possible to partition $V(G)$ into r parts such that each part contains exactly one of the terminals of T and there are at most ℓ edges between different parts (i.e., at most ℓ crossing edges). In the special case where $|T| = 2$, this gives the celebrated Minimum Cut problem, which is polynomially solvable [31]. In general, when there is no

* The third author was co-financed by the European Union (European Social Fund ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF), Research Funding Program: ARISTEIA II.

† A complete version of this extended abstract has been appeared at http://arxiv.org/abs/1509.07404

10th International Symposium on Parameterized and Exact Computation (IPEC 2015).
Editors: Thore Husfeldt and Iyad Kanj; pp. 78–89

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

restriction on the number of terminals, the Multiway Cut problem is NP-complete [8] and a lot of research has been devoted to the study of this problem and its generalizations, including several classic results on its polynomial approximability [18, 3, 14, 17, 24, 30].

More recently, special attention to the Multiway Cut problem was given from the parameterized complexity point of view. The existence of an FPT-algorithm for Multiway Cut (when parameterized by ℓ), i.e., an $f(\ell) \cdot n^{O(1)}$-step algorithm, had been a long-standing open problem. This question was answered positively by Marx in [26] with the use of the *important separators technique* which was also used for the design of FPT-algorithms for several other problems such as Directed Multiway Cut [4], Vertex Multicut, and Edge Multicut [28]. This technique has been extended to the powerful framework of *randomized contractions technique*, introduced in [5]. This made it possible to design FPT-algorithms for several other problems such as Unique Label Cover, Steiner Cut, Edge/Vertex Multiway Cut-Uncut. We stress that this technique is quite versatile.

In this paper we use it in order to design FPT-algorithms for parameterizations of two problems that do not seem to be directly related to each other: the Min-Max-Multiway Cut problem [32] and the List Digraph Homomorphism problem.

1.1 Min-Max-Multiway Cut

In the Multiway Cut problem the parameter ℓ bounds the total number of crossing edges (i.e., edges with endpoints in different parts). Svitkina and Tardos [32] considered a "min-max" variant of this problem, namely the Min-Max-Multiway Cut, where ℓ bounds the maximum number of outgoing edges of the parts[1]. In [32], it was proved that Min-Max-Multiway Cut is NP-complete even when the number of terminals is $r = 4$. As a consequence of the results in [32] and [29], Min-Max-Multiway Cut admits an $O(\log^2 n)$-approximation algorithm. This was improved recently in [1] to a $O((\log n \cdot \log r)^{1/2})$-approximation algorithm.

To our knowledge, nothing is known about the parameterized complexity of this problem. We prove the following.

▶ **Theorem 1.** *There exists an algorithm that solves the* Min-Max-Multiway Cut *problem in* $2^{O((r\ell)^2 \log r\ell)} \cdot n^4 \cdot \log n$ *steps, i.e.,* Min-Max-Multiway Cut *belongs to FPT when parameterized by both* r *and* ℓ.

(Throughout the paper, we use $n = |V(G)|$ when we refer to the number of vertices of the graph G in the instance of the considered problem.)

1.2 List Digraph Homomorphism

Given two directed graphs G and H, an *H-homomorphism* of G is a mapping $\chi : V(G) \to V(H)$ such that if (x, y) is an arc of G, then $(\chi(x), \chi(y))$ is also an arc in H. In the List Digraph Homomorphism problem, we are given two graphs G and H and a list function $\lambda : V(G) \to 2^{V(H)}$ and we ask whether G has a H-homomorphism such that for every vertex v of G, $\chi(v) \in \lambda(v)$. Graph and digraph homomorphisms have been extensively studied both from the combinatorial and the algorithmic point of view (see e.g., [21, 2, 13, 15, 16]).

Especially for the List Digraph Homomorphism problem, a dichotomy characterizing the instantiations of H for which the problem is hard was given in [22] (see also [12]). Notice that the standard parameterization of List Digraph Homomorphism by the size of the

[1] Notice that under this viewpoint Multiway Cut can be seen as Min-Sum-Multiway Cut.

graph H is para-NP-complete as it yields the 3-Coloring problem when G is restricted to be a simple graph and $H = K_3$. A more promising parameterization of List Homomorphism (for undirected graphs) has been introduced in [10], where the parameter is a bound on the number of pre-images of some prescribed set of vertices of H (see also [9, 11, 27]). Another parameterization, again for the undirected case, was introduced in [6], where the parameter is the number of vertices to be removed from the graph G so that the remaining graph has a list H-homomorphism.

We introduce a new parameterization of List Digraph Homomorphism where the parameter is, apart from $h = |V(H)|$, the number of "crossing edges", i.e., the edges of G whose endpoints are mapped to different vertices of H. For this, we enhance the input with an integer ℓ and ask for a list digraph homomorphism with at most ℓ crossing edges. Clearly, when $\ell = |E(G)|$, this yields the original problem. We call the new problem Bounded List Digraph Homomorphism (in short, BLDH). Notice that the fact that List Digraph Homomorphism is NP-complete even when $h = 3$, implies that BLDH is para-NP-complete when parameterized only by h. The input of BLDH is a quadruple (G, H, λ, ℓ) where G is the guest graph, H is the host graph, $\lambda : V(G) \to 2^{V(H)}$ is the list function and ℓ is a non-negative integer. Our next step is to observe that BLDH is W[1]-hard, when parameterized only by ℓ. To see this consider an input (G, k) of the Clique problem and construct the input $(K, \bar{G}, \lambda, \ell)$ where K is a the complete digraph on k vertices, $\bar{G}$ is the digraph obtained by G by replacing each edge by two opposite direction arcs between the same endpoints, $\lambda = \{(v, V(G)) \mid v \in V(K)\}$, and $\ell = k(k-1)$. Notice that (G, k) is a yes-instance of Clique iff $(K, \bar{G}, \lambda, \ell)$ is a yes-instance of BLDH.

We conclude that when BLDH is parameterized by ℓ or h only, then one may not expect it to be fixed parameter tractable. This means that the parameterization of BLDH by h and ℓ is meaningful to consider. Our result is the following.

▶ **Theorem 2.** *There exists an algorithm that solves the* Bounded List Digraph Homomorphism *problem in* $2^{O(\ell \cdot \log h + \ell^2 \cdot \log \ell)} \cdot n^4 \cdot \log n$ *steps, i.e.,* Bounded List Digraph Homomorphism *belongs to FPT when parameterized by the number ℓ of crossing edges and the number h of vertices of H.*

1.3 List Allocation

In order to prove Theorems 1 and 2, we prove that both problems are Turing FPT-reducible[2] to a single new problem that we call List Allocation (in short, LA).

The List Allocation problem is defined as follows: We are given a graph G and a set of r "boxes" indexed by numbers from $\{1, \ldots, r\}$. Each vertex v of G is accompanied with a list $\lambda(v)$ of indices corresponding to the boxes where it is allowed to be allocated. Moreover, there is a weight function α assigning to every pair of different boxes a non-negative integer. The question is whether there is a way to place each of the vertices of G into some box of its list such that, for any two different boxes i and j, the number of crossing edges between them is *exactly* $\alpha(i, j)$.

As we easily see in Subsection 2.3, List Allocation is NP-complete, even when $r = 2$. Throughout this paper, we parameterize the List Allocation problem by the total number w of "crossing edges" between different boxes, i.e., $w = \sum_{1 \le i < j \le r} \alpha(i, j)$.

[2] Let **A** and **B** be two parameterized problems. We say that a parameterized problem **A** is *Turing FPT-reducible* to **B** when the existence of an FPT-algorithm for **B** implies the existence of an FPT-algorithm for **A**. (For brevity, in this paper, we write "T-FPT" instead of "Turing FPT".)

Our main result is that this parameterization of LA is in FPT (the basic ideas of the algorithm are given in Section 4.

▶ **Theorem 3.** *There exists an algorithm that, given as input an instance* $I = (G, r, \lambda, \alpha)$ *of* List Allocation, *returns an answer to this problem in* $2^{O(w^2 \cdot \log w)} \cdot n^4 \cdot \log n$ *steps, where* $w = \sum_{1 \leq i < j \leq r} \alpha(i, j)$.

To witness the expressive power of List Allocation, let us first exemplify why Multiway Cut, parameterized by w, is T-FPT-reducible to List Allocation. Given an instance of Multiway Cut, we first discard from its graph all the connected components that have at most 1 terminal. Clearly, this gives an equivalent instance $(G, T = \{t_1, \ldots, t_r\}, w)$ where $r \leq w + 1$.

Next, we consider the set $\mathcal{A}$ containing every weight function α such that $\sum_{1 \leq i < j \leq r} \alpha(i, j) \leq w$. Let also $\lambda : V(G) \to 2^{[r]}$ be the list function such that if $v = t_i \in T$, then $\lambda(v) = \{i\}$, otherwise $\lambda(v) = \{1, \ldots, r\}$. It is easy to verify that (G, T, w) is a yes-instance of Multiway Cut if and only if there exists some $\alpha \in \mathcal{A}$ such that (G, r, λ, α) is a yes-instance of List Allocation. This yields the claimed reduction, as $|\mathcal{A}|$ is clearly bounded by some function of w. This reduction to the List Allocation problem turns out to be quite flexible and, as we will see in Subsection 3.1 (Theorem 4), it can easily be adapted to a T-FPT-reduction of Min-Max-Multiway Cut to List Allocation. The reduction of Bounded List Digraph Homomorphism to List Allocation is more complicated and is described in Subsection 3.2 (Theorem 10). Theorem 3, together with the aforementioned reductions, yields Theorems 1 and 2.

2 Preliminaries and the definition of List Allocation

2.1 Functions and allocations

We use the notation $\log(n)$ to denote $\lceil \log_2(n) \rceil$ for $n \in \mathbb{Z}_{\geq 1}$ and we agree that $\log(0) = 1$. Given a non-negative integer n, we denote by $[n]$ the set of all positive integers no bigger than n. Given a finite set A and an integer $s \in \mathbb{Z}_{\geq 0}$, we denote by $\binom{A}{s}$ (resp. $\binom{A}{\leq s}$) the set of all subsets of A with exactly (resp. at most) s elements. Given a function $f : A \to \mathbb{Z}_{\geq 0}$ we define $\sum f = \sum_{x \in A} f(x)$. An *$r$-allocation* of a set S is an r-tuple $\mathcal{V} = (V_1, \ldots, V_r)$ of, possibly empty, sets that are pairwise disjoint and whose union is the set S. We refer to the elements of $\mathcal{V}$ as the *parts* of $\mathcal{V}$ and we denote by $\mathcal{V}^{(i)}$ the i-th part of $\mathcal{V}$, i.e., $\mathcal{V}^{(i)} = V_i$.

2.2 Definitions about graphs

In this paper, when giving the running time of an algorithm of some problem whose instance involves a graph G, we agree that $n = |V(G)|$ and $m = |E(G)|$.

All graphs in this paper are loopless and they may have multiple edges. The only exception to this agreement is in Subsection 3.2 where we also allow loops. If G is a graph and X, Y are two disjoint vertex subsets of $V(G)$, we define $\delta_G(X, Y)$ as the set of edges with one endpoint in X and the other in Y. Given a graph G, denote by $\mathcal{C}(G)$ the collection of all connected components of G.

2.3 The list allocation problem

We define the problem LA as follows.

List Allocation (LA)
Input: A tuple $I = (G, r, \lambda, \alpha)$ where G is a graph, $r \in \mathbb{Z}_{\geq 1}$, $\lambda : V(G) \to 2^{[r]}$, and $\alpha : \binom{[r]}{2} \to \mathbb{Z}_{\geq 0}$.
Output: An r-allocation $\mathcal{V}$ of $V(G)$ such that
1. $\forall \{i,j\} \in \binom{[r]}{2}$, $\lvert\delta_G(\mathcal{V}^{(i)}, \mathcal{V}^{(j)})\rvert = \alpha(i,j)$ and
2. $\forall v \in V(G), \forall i \in [r]$, if $v \in \mathcal{V}^{(i)}$ then $i \in \lambda(v)$,
or a correct report that no such r-allocation exists.

For simplicity, in the above definition we write $\alpha(\{i,j\})$ as $\alpha(i,j)$ and we agree that $\alpha(i,j) = \alpha(j,i)$. Also, given an instance I of LA, we denote[3] $w(I) = \sum \alpha$. We will also use w instead of $w(I)$ when it is clear what is the instance we are working with. We assume that the multiplicity of each edge in G does not exceed w as, if this happens, then reducing it to w creates an equivalent instance of the problem.

In the definition of LA each vertex v of G carries a *list* $\lambda(v)$ indicating the parts where v can be possibly allocated. Moreover, α is a function assigning weights to pairs of parts in $\mathcal{V}$. The weights defined by α prescribe the precise number of crossing edges between distinct parts of $\mathcal{V}$.

Notice that LA is an NP-hard problem by a simple reduction from the Max Cut problem, asking whether, for an input graph G and some $w \in \mathbb{Z}_{\geq 0}$, whether there is a partition V_1, V_2 of $V(G)$ such that there are *exactly*[4] w edges each with endpoints in both V_1 and V_2. Indeed, given an instance $I = (G, w)$ of Max Cut, construct the instance $I' = (G, 2, \lambda, \alpha)$ where $\lambda(v) = \{1, 2\}$ for every $v \in V(G)$ and $\alpha(1,2) = w$. Note also that when $r = 2$, LA is polynomially solvable on planar graphs as it directly reduces to Planar Max Cut that is polynomially solvable [20].

3 Main reductions

In this section we formally define Min-Max-Multiway Cut and List Digraph Homomorphism and we reduce them to List Allocation.

3.1 Min-Max-Multiway Cut

The Min-Max-Multiway Cut problem is formally defined as follows:

Min-Max-Multiway Cut
Input: A tuple $I = (G, \ell, r, T)$ where G is an undirected graph, $\ell, r \in \mathbb{Z}_{\geq 0}$, and $T \subseteq V(G)$ with $\lvert T\rvert = r$.
Output: A partition $\{\mathcal{P}_1, \ldots, \mathcal{P}_r\}$ of $V(G)$ such that for every $i \in [r]$, it holds that $\lvert\mathcal{P}_i \cap T\rvert = 1$ and $\lvert\delta_G(\mathcal{P}_i, V(G) \setminus \mathcal{P}_i)\rvert \leq \ell$, or a correct report that no such partition exists.

Similarly to the case of LA, we assume that the multiplicity of each edge in G does not exceed ℓ.

▶ **Theorem 4.** *If there is an algorithm that solves* LA *in $T(n, w(I))$ steps, then there exists an algorithm that solves* Min-Max-Multiway Cut *in $2^{O(r \cdot \min\{\ell \cdot \log r, r \cdot \log \ell\})} \cdot T(n, r\ell)$ steps.*

[3] Given a function $\tau : A \to \mathbb{Z}_{\geq 0}$, we denote $\sum \tau = \sum_{x \in A} \tau(x)$.

[4] It is straightforward to see that the standard reduction from Nae-3-Sat also works when the question of Max Cut asks for exactly w crossing edges instead of at least w crossing edges.

Proof. Given an input $I = (G, \ell, r, T)$ of Min-Max-Multiway Cut, we fix (arbitrarily) a bijection $\mu : V(T) \to [r]$ and we define $\lambda : V(G) \to 2^{[r]}$ such that

$$\lambda(x) = \begin{cases} [r] & \text{if } x \in V(G) \setminus T \\ \{\mu(x)\} & \text{if } x \in T. \end{cases}$$

We now consider the family $\mathcal{U}(I)$ of instances of LA containing one element $I' = (G, r, \lambda, \alpha)$ for each choice of function $\alpha : \binom{[r]}{2} \to \mathbb{Z}_{\geq 0}$ satisfying

$$\forall i \in [r], \sum_{j \in [r] \setminus i} \alpha(i, j) \leq \ell.$$

Notice that I is a yes-instance of Min-Max-Multiway Cut if and only if there exists some $I' \in \mathcal{U}(I)$ that is a yes-instance of LA. As $|\mathcal{U}(I)| = 2^{O(r \cdot \min\{\ell \cdot \log r, r \cdot \log \ell\})}$ and for each $I' \in \mathcal{U}(I)$ it holds that $w(I') = O(r\ell)$, the result follows. ◀

3.2 List Digraph Homomorphism

Let G and H be directed graphs where G is simple and H may have loops but not multiple directed edges. A (directed) edge in the digraph G from the vertex x to the vertex y is denoted by (x, y). Let also $\lambda : V(G) \to 2^{V(H)}$. We denote by $E_1(H)$ the loops of H and by $E_2(H)$ the edges of H between distinct vertices. An *λ-list H-homomorphism* of G is a function $\chi : V(G) \to V(H)$ such that

- $\chi(v) \in \lambda(v)$ for every $v \in V(G)$, and
- $(\chi(u), \chi(v)) \in E(H)$ for every $(u, v) \in E(G)$.

Given a list H-homomorphism χ of G and an edge $e = (a, b) \in E_2(H)$ we define

$$C(e) = \{(u, v) \in E(G) \mid \chi(u) = a \text{ and } \chi(v) = b\}.$$

Bounded List Digraph Homomorphism is formally defined as follows.

Bounded List Digraph Homomorphism (BLDH)
Input: A tuple $I = (G, H, \lambda, \ell)$ where G and H are digraphs, $\lambda : V(G) \to 2^{V(H)}$, and $\ell \in \mathbb{N}_{\geq 0}$.
Output: A *λ-list H-homomorphism* of G where $\sum_{e \in E(H)} \lvert C(e) \rvert \leq \ell$ or a correct report that no such homomorphism exists.

We now define the following more general problem.

Arc-Specified List Digraph Homomorphism (ASLDH)
Input: A tuple $I = (G, H, \lambda, \alpha)$ where G and H are digraphs, $\lambda : V(G) \to 2^{V(H)}$, and $\alpha : E_2(H) \to \mathbb{Z}_{\geq 0}$.
Output: A λ-list H-homomorphism χ of G such that $\forall_{e \in E_2(H)} \lvert C(e) \rvert = \alpha(e)$ or a correct report that no such λ-list H-homomorphism exists.

Given an instance $I = (G, H, \lambda, \alpha)$ of ASLDH we define $d(I) = \sum \alpha$. As we already did for the cases of LA and Min-Max-Multiway Cut, we assume that the multiplicity of the edges of the instance of BLDH (resp. ASLDH) does not exceed ℓ (resp. $d(I)$).

In the next sections we will prove that there exists an FPT-algorithm for ASLDH, when parameterized by *both* $h = |V(H)|$ and $d = d(I)$. This fact together with the following result yields Theorem 2.

▶ **Theorem 5.** *If there is an algorithm that solves* ASLDH *in $T(n, d(I))$ steps, then there exists an algorithm that solves* BLDH *in $2^{O(\ell \log h)} \cdot T(n, \ell)$ steps where $h = |V(H)|$.*

Proof. Given an instance $I = (G, H, \lambda, \ell)$ of BLDH we set $\mathcal{U}(I) = \{(G, H, \lambda, \alpha) \mid \sum \alpha \leq \ell\}$ and we observe that I is a YES-instance of BLDH if and only if some $I' \in \mathcal{U}(I)$ is a YES-instance of ASLDH. The lemma follows as $|\mathcal{U}(I)| = 2^{O(\ell \log h)}$ and $d(I') \leq \ell$. ◀

3.3 A sparsifier for ASLDH

In order to prove that ASLDH admits an FPT-algorithm when parameterized by *both* $h = |V(H)|$ and $d = d(I)$, we will give a Turing-FPT reduction of ASLDH to LA in Subsection 3.4. The latter problem can be solved by an FPT-algorithm due to the result of Section 4. The reduction of Subsection 3.4 receives an instance (G, H, λ, α) of ASLDH and returns an equivalent instance $(G', r, \lambda', \alpha')$ of LA where $|V(G')| = O(|E(G)|)$ which is $O(w \cdot |V(G)|^2)$, in general. In order to avoid this blow-up in the polynomial running time of our final FPT-algorithm, we give a way to transform the instances of ASLDH to equivalent instances of the same problem whose graphs are sparse. This "sparsification" procedure is described below.

A graph is *d-edge connected* if it has at least two vertices and for every two vertices there are d edge disjoint paths between them. We use the following result from [25].

▶ **Proposition 6.** *For every $d \in \mathbb{Z}_{\geq 1}$, every graph G where $|E(G)| \geq d \cdot (|V(G)| - 1)$ contains a d-edge connected subgraph.*

We also need the following result.

▶ **Lemma 7.** *Let G be a graph and let $\mathcal{C} = \{C_1, \ldots, C_r\}$ be a collection of vertex disjoint connected subgraphs of G. Let also G' be the graph obtained if we contract in G all edges in the graphs in $\mathcal{C}$. If G' is d-edge connected and each graph in $\mathcal{C}$ is d-edge connected or a single vertex, then G contains a subgraph that is d-edge connected.*

Given a graph H and a positive integer d, we say that a subgraph H of G is a *d-edge connected core of G* if every connected component of H is d-edge connected and, among all such subgraphs of G, H has maximum number of edges. The proof of the next lemma uses Proposition 6.

▶ **Lemma 8.** *For every $d \in \mathbb{Z}_{>0}$, every graph G with $m \geq d \cdot (n - 1)$ contains a unique d-edge connected core that can be found in $O(d \cdot n^4)$ steps.*

Proof. The claimed d-edge connected core exists because of Proposition 6. Also, it is unique because if there are two d-edge connected cores J_1 and J_2, then it can be easily checked that the graph $J_1 \cup J_2$ is also a d-edge connected core of G. The algorithm repetitively removes from G edges of min-cuts of size at most $d - 1$ in its connected components (each can be found in $O(d \cdot n^3)$ steps according to [31]) until this is not possible anymore (isolated vertices, when appearing during this procedure, are removed).

Note that the total number of steps of this procedure is bounded by the running time of the algorithm in [31] times the number of connected components of the resulting graph. This justifies the claimed running time. Let J be the d-edge connected core of G. Notice that none of the edges of J will be deleted by this procedure. Indeed, assuming the opposite, let G' be the graph where for the first time a cut (V_1, V_2) is found where the set F of crossing edges contains some edge $e = \{x, y\}$ in J. Let also C be the connected component of G' containing this cut and let C_J be a connected component of J that is a subgraph of C containing e.

Notice that x and y belong to different connected components of $C \setminus F$ and therefore also to different connected components of $C_J \setminus F$, contradicting the fact that C_J is d-edge connected. We just proved that the output of the algorithm will be a subgraph of J. Notice also that each connected component of this output is d-edge connected. By the maximality of J, this output is necessarily J. ◀

▶ **Lemma 9.** *There is an $O(d(I) \cdot n^4)$-step algorithm that given in instance $I = (G, H, \lambda, \alpha)$ of* ASLDH*, outputs an equivalent instance $I' = (G', H, \lambda', \alpha)$ of the same problem where $|E(G')| = O(d(I) \cdot |V(G')|)$.*

Proof. Let $\tilde{G}$ be the underlying graph of G (multiplicities of edges of opposite direction are summed up) and $d = d(I)$. If $\tilde{G}$ does not contains a $(d+1)$-edge connected core, then, from Proposition 6, $|E(G)| = O(d \cdot |V(G)|)$.

Suppose now that $\tilde{G}$ has a $(d+1)$-edge connected core J that, from Lemma 8, can be found in $O(d \cdot |V(G)|^4)$ steps. We create a new graph G' as follows: for each $C \in \mathcal{C}(J)$ we contract all vertices of C to a single vertex v_C and we update λ to λ' so that if $x \notin \{v_C \mid C \in \mathcal{C}(J)\}$, then $\lambda'(x) = \lambda(x)$ and if $x = v_C$, then $\lambda'(x) = \cap_{y \in V(C)} \lambda(y)$. We claim that $I' = (G', H, \lambda', \alpha)$ is an equivalent instance of ASLDH. Indeed, this is based on the fact that, given a λ-list H-homomorphism χ of G and a connected component C of J, all vertices of J should be the preimages via χ of the same vertex of H. To verify this fact, just observe that, if this is not the case, then the removal of the $\leq d$ crossing edges from C (i.e., edges with endpoints mapped to different vertices of H) will disconnect C, a contradiction to the $(d+1)$-edge-connectivity of C.

It now remains to prove that $|E(G')| = O(d \cdot |V(G')|)$. If $|E(G')| \geq (d+1) \cdot (|V(G)'| - 1)$, then, again from Proposition 6, G' contains a $(d+1)$-edge connected subgraph. This, because of Lemma 7, implies that G contains a subgraph that is $(d+1)$-edge connected and has more edges than J, a contradiction. ◀

3.4 A reduction of ASLDH to LA

Given the results of the previous section we are now in position to prove the following.

▶ **Theorem 10.** *If there is an algorithm that solves* LA *in $T(n, w(I))$ steps, then there exists an algorithm that solves* ASLDH *in $T\big(O(d(I) \cdot n), O(d(I))\big) + O(d(I) \cdot n^4)$ steps.*

Proof. Let $I = (G, H, \lambda, \alpha)$ be an instance of ASLDH. Using the algorithm of Lemma 9, we may assume that $|E(G)| = O(d(I) \cdot |V(G)|)$. We then use I to generate an instance $I' = (G', r, \lambda', \alpha')$ of LA, as follows:

- $G' = (V', E')$, where
  - $V' = V \cup V_F \cup V_L$, where $V = V(G)$, $V_F = \{f_{uv} \mid (u, v) \in E(G)\}$, and $V_L = \{\ell_{uv} \mid (u, v) \in E(G)\}$ and
  - $E' = E \cup E_F \cup E_L$, where $E = \{\{f_{uv}, \ell_{uv}\} \mid (u, v) \in E(G)\}$, $E_F = \{\{u, f_{uv}\} \mid (u, v) \in E(G)\}$, $E_L = \{\{\ell_{uv}, v\} \mid (u, v) \in E(G)\}$.
- $r = |V(H)| + 2 \cdot |E_2(H)|$ and $\sigma : V(\tilde{H}) \to [r]$ is a bijection where $\tilde{H}$ is the graph obtained from H by subdividing twice each of its arcs that are not loops. For each arc $(x, y) \in E_2(H)$, we denote its corresponding path in $\tilde{H}$ as P_{xy}, where $V(P_{xy}) = \{x, \tilde{f}_{xy}, \tilde{\ell}_{xy}, y\}$.

- $\lambda' : V(G') \to [r]$ such that

$$\lambda'(w) = \begin{cases} \{\sigma(x) \mid x \in \lambda(w)\} & \text{if } w \in V \\ \{\sigma(\tilde{f}_{xy}) \mid x \in \lambda(u) \wedge y \in \lambda(v) \wedge x \neq y\} \cup \\ \{\sigma(x) \mid x \in \lambda(u) \cap \lambda(v) \wedge (x,x) \in E_1(H)\} & \text{if } w = f_{uv} \in V_F \\ \{\sigma(\tilde{\ell}_{xy}) \mid x \in \lambda(u) \wedge y \in \lambda(v) \wedge x \neq y\} \cup \\ \{\sigma(x) \mid x \in \lambda(u) \cap \lambda(v) \wedge (x,x) \in E_1(H)\} & \text{if } w = \ell_{uv} \in V_L. \end{cases}$$

- $\alpha' : \binom{[r]}{2} \to \mathbb{Z}_{\geq 0}$ such that

$$\alpha'(i,j) = \begin{cases} \alpha(x,y) & \text{if there exists some } (x,y) \in E_2(H) \text{ such that} \\ & (i,j) \in \big\{(\sigma(x),\sigma(\tilde{f}_{xy})),(\sigma(\tilde{f}_{xy})),\sigma(\tilde{\ell}_{xy})),(\sigma(\tilde{\ell}_{xy}),\sigma(y))\big\} \\ 0 & \text{otherwise.} \end{cases}$$

Let $\chi : V(G) \to V(H)$ be a λ-list H-homomorphism of G where $\forall_{e \in E_2(H)} |C(e)| = \alpha(e)$. We construct an r-allocation $\mathcal{V}$ of $V(G')$ as follows:

- for every $u \in V = V(G)$, u belongs to the part $\mathcal{V}^i$, where $i = \sigma(\chi(u))$
- for every $f_{uv} \in V_F$, f_{uv} belongs to the part $\mathcal{V}^i$, where

$$i = \begin{cases} \sigma(\chi(u)) & \text{if } \chi(u) = \chi(v) \\ \sigma(\tilde{f}_{xy}) & \text{if } x = \chi(u) \neq y = \chi(v) \end{cases}$$

- for every $\ell_{uv} \in V_L$, ℓ_{uv} belongs to the part $\mathcal{V}^i$, where

$$i = \begin{cases} \sigma(\chi(u)) & \text{if } \chi(u) = \chi(v) \\ \sigma(\tilde{\ell}_{xy}) & \text{if } x = \chi(u) \neq y = \chi(v) \end{cases}$$

It is easy to verify that $\mathcal{V}$ is a solution for I'.

Now consider a solution $\mathcal{V}$ for I'. From $\mathcal{V}$, we define a mapping $\chi : V(G) \to V(H)$ so that for every $u \in V$, we have that $\chi(u) = \sigma^{-1}(i)$ if and only if $u \in \mathcal{V}^{(i)}$. We claim that χ is a λ-list H-homomorphism of G where $\forall_{e \in E_2(H)} |C(e)| = \alpha(e)$. For this, we investigate χ upon two conditions: firstly, we verify that χ is a λ-list H-homomorphism, and secondly that $\forall_{e \in E_2(H)} |C(e)| = \alpha(e)$.

Let us prove that χ is a λ-list H-homomorphism. To see that $\chi(u) \in \lambda(u)$ for every $u \in V(G)$, let u be in the i-th part of $\mathcal{V}$. Since $i \in \lambda'(u)$, the construction of λ' implies that $\sigma^{-1}(i) \in \lambda(u)$, and thus $\chi(u) \in \lambda(u)$. To see that χ is an H-homomorphism, for an arbitrary edge $(u,v) \in E(G)$ we shall show that $(\chi(u),\chi(v)) \in E_1(H) \cup E_2(H)$. Let u and v respectively belong to $\sigma(x)$-th and $\sigma(y)$-th parts of $\mathcal{V}$, for some $x, y \in V(\tilde{H})$. Note that $x \in \lambda(u) \subseteq V(H)$ and $y \in \lambda(v) \subseteq V(H)$. There are two possibilities: $x \neq y$ or $x = y$.

Case 1: $x \neq y$**.** Since σ is a bijection, this means $\sigma(x) \neq \sigma(y)$. From the way we construct α', the vertices f_{uv} and ℓ_{uv} can be only allocated into the $\sigma(\tilde{f}_{xy})$-th part and the $\sigma(\tilde{\ell}_{xy})$-th part, respectively, in the solution $\mathcal{V}$. Furthermore, the construction of α' also implies $(x,y) \in E_2(H)$.

Case 2: $x = y$. This means $\sigma(x) = \sigma(y)$. The construction of α' implies f_{uv} and ℓ_{uv} are allocated into the $\sigma(x)$-th part of $\mathcal{V}$ as well. This, in turn, means that $\sigma(x) \in \lambda'(f_{uv})$ and $\sigma(x) \in \lambda'(\ell_{uv})$. Recall that $\lambda'(f_{uv})$ contains $\sigma(x)$ only when $(x, x) \in E_1(H)$. Hence, $(x, y) \in E_1(H)$.

Now we verify that $\forall_{e \in E_2(H)} |C(e)| = \alpha(e)$. Consider an arc $e = (x, y) \in E_2(H)$. Note that for every directed edge (u, v) in the χ-arc charge $C(e)$, the (u, f_{uv}) of $E(G')$ contributes to $\alpha'(\sigma(x), \sigma(\tilde{f}_{xy}))$ exactly by one unit. Conversely, for every edge (u, f_{uv}) of $E(G')$ which contributes to $\alpha'(\sigma(x), \sigma(\tilde{f}_{xy}))$, we have $\chi(v) = y$ and thus the directed arc (u, v) contributes to $C(e)$ by one unit. This establishes that $\forall_{e \in E_2(H)} |C(e)| = \alpha(e)$.

The claimed running time follows from the fact that $w(I') = \sum \alpha' = 3 \cdot \sum \alpha = O(d(I))$ and $|V(G')| = O(|E(G)|) = O(d(I) \cdot |V(G)|)$. ◀

4 An FPT-algorithm for List Allocation

In this section we give a brief description of the T-FPT-reductions required to prove that LA admits an FPT-algorithm, i.e., the proof of Theorem 3. This is the most technical part of our paper. Below we summarize the main steps.

1. List Allocation is T-FPT-reduced to its restriction, called CLA, where G is a connected graph and only $O(w)$ boxes are used. This reduction takes care of the different ways connected components of G can entirely be placed into the boxes and is based on dynamic programming.
2. CLA is T-FPT-reduced to a restriction of it, called HCLA, where G is highly connected in the sense that there is no set of w edges that can separate G into two "big" connected components. This reduction uses the technique of *recursive understanding*, introduced in [23] and further developed in [7] and [5] (see also [19]), for generalizations of the Multiway Cut problem).
3. HCLA is T-FPT-reduced to a special enhancement of it, called S-HCLA, whose input additionally contains some set $S \subseteq V(G)$ and the problem asks for a solution where all vertices of S are placed in a unique "big" box and all vertices of this box which are incident to crossing edges are contained in S. This variant of the problem permits the application of the technique of *randomized contractions*, introduced in [5].
4. Finally, S-HCLA is T-FPT-reduced to List Allocation restricted to instances whose sizes are bounded by a function of the parameter. It is a dynamic programming based on the fact that an essentially equivalent instance of the problem can be constructed if, apart from S, we remove from G all but a bounded number of the connected components of $G \setminus S$.

5 Further research

In the definition of List Allocation we ask for a *λ-list H-homomorphism* of G where $\sum_{e \in E(H)} |C(e)| \leq \ell$. A different parameterization of List Allocation, that is similar in flavor to Min-Max Multiway Cut, may instead ask for a *λ-list H-homomorphism* of G where $\max_{v \in V(H)} \sum_{e \text{ is incident to } v} |C(e)| \leq \ell$. We call this new problem Max Bounded List Digraph Homomorphism (in short MBLDH) As it is straightforward to prove an analogue of Theorem 5, where BLDH is now replaced by MBLDH and instead of $2^{O(\ell \log h)} \cdot T(n, \ell)$ steps we now have a reduction that takes $2^{O(\ell^2 \log h)} \cdot T(n, \ell)$ steps. This

implies that MBLDH, when parameterized by ℓ and h admits an FPT-algorithm that runs in $2^{O(\ell^2 \cdot \max\{\log \ell, \log h\})} \cdot n^4 \cdot \log n$ steps.

A natural research direction is to improve the running time of our FPT-algorithms for Min-Max Multiway Cut and Bounded List Digraph Homomorphism. If we want to improve our running times using the techniques used in this paper it seems that we need to crucially improve upon the recursive understanding and randomized contractions technique.

Acknowledgement. We would like to thank the anonymous referees of an earlier version of this paper for their thorough remarks and suggestions that improved the presentation and some proofs of the paper.

References

1 Nikhil Bansal, Uriel Feige, Robert Krauthgamer, Konstantin Makarychev, Viswanath Nagarajan, Joseph (Seffi) Naor, and Roy Schwartz. Min-max graph partitioning and small set expansion. In *Proc. of the 52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 17–26. IEEE Computer Society, 2011.

2 Ronald Brown, Ifor Morris, J. Shrimpton, and Christopher D. Wensley. Graphs of morphisms of graphs. *Electronic Journal of Combinatorics*, 15(1), 2008.

3 Chandra Chekuri, Sudipto Guha, and Joseph Naor. The steiner k-cut problem. *SIAM Journal on Discrete Mathematics*, 20(1):261–271, 2006.

4 Rajesh Chitnis, MohammadTaghi Hajiaghayi, and Dániel Marx. Fixed-parameter tractability of directed multiway cut parameterized by the size of the cutset. In *Proc. of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1713–1725. SIAM, 2012.

5 Rajesh Hemant Chitnis, Marek Cygan, MohammadTaghi Hajiaghayi, Marcin Pilipczuk, and Michal Pilipczuk. Designing FPT algorithms for cut problems using randomized contractions. In *Proc. of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 460–469. IEEE Computer Society, 2012.

6 Rajesh Hemant Chitnis, László Egri, and Dániel Marx. List H-coloring a graph by removing few vertices. In *Proc. of the 21st Annual European Symposium on Algorithms (ESA)*, volume 8125 of *LNCS*, pages 313–324, 2013.

7 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, MichałPilipczuk, and Saket Saurabh. Minimum bisection is fixed parameter tractable. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC)*, pages 323–332. ACM, 2014.

8 E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, (4):864–894, 1994.

9 Josep Díaz, Maria Serna, and Dimitrios M. Thilikos. Efficient algorithms for counting parameterized list H-colorings. *Journal of Computer and System Sciences*, 74(5):919–937, 2008.

10 Josep Díaz, Maria J. Serna, and Dimitrios M. Thilikos. (H, C, K)-coloring: Fast, easy, and hard cases. In *Proc. of the 26th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 2136 of *LNCS*, pages 304–315, 2001.

11 Josep Díaz, Maria J. Serna, and Dimitrios M. Thilikos. Fixed parameter algorithms for counting and deciding bounded restrictive list h-colorings. In *Proc. of the 12th Annual European Symposium on Algorithms (ESA)*, volume 3221 of *LNCS*, pages 275–286, 2004.

12 László Egri, Pavol Hell, Benoit Larose, and Arash Rafiey. Space complexity of list H-colouring: a dichotomy. In *Proc. of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 349–365. SIAM, 2014.

13 László Egri, Andrei Krokhin, Benoit Larose, and Pascal Tesson. The complexity of the list homomorphism problem for graphs. *Theory of Computing Systems*, 51(2):143–178, 2012.

14 Guy Even, Joseph Naor, Baruch Schieber, and Madhu Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2):151–174, 1998.

15 Tomás Feder, Pavol Hell, and Jing Huang. List homomorphisms and circular arc graphs. *Combinatorica*, 19(4):487–505, 1999.

16 Tomás Feder, Pavol Hell, and Jing Huang. Bi-arc graphs and the complexity of list homomorphisms. *Journal of Graph Theory*, 42(1):61–80, 2003.

17 Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Multiway cuts in node weighted graphs. *Journal of Algorithms*, 50(1):49–61, 2004.

18 Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995.

19 Martin Grohe, Ken-ichi Kawarabayashi, Dániel Marx, and Paul Wollan. Finding topological subgraphs is fixed-parameter tractable. In *Proc. of the 43rd ACM Symposium on Theory of Computing (STOC)*, pages 479–488. ACM, 2011.

20 F. Hadlock. Finding a maximum cut of a planar graph in polynomial time. *SIAM Journal on Computing*, 4(3):221–225, 1975.

21 Pavol Hell and Jaroslav Nešetřil. *Graphs and homomorphisms*, volume 28 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, Oxford, 2004.

22 Pavol Hell and Arash Rafiey. The dichotomy of list homomorphisms for digraphs. In *Proc. of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1703–1713. SIAM, 2011.

23 Ken ichi Kawarabayashi and Mikkel Thorup. The minimum k-way cut of bounded size is fixed-parameter tractable. In *Proc. of the 54th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 160–169. IEEE Computer Society, 2013.

24 David R. Karger, Philip Klein, Cliff Stein, Mikkel Thorup, and Neal E. Young. Rounding algorithms for a geometric embedding of minimum multiway cut. In *Proc. of the 31st Annual ACM Symposium on Theory of Computing (STOC)*, pages 668–678. ACM, 1999.

25 Lefteris M. Kirousis, Maria Serna, and Paul Spirakis. Parallel complexity of the connected subgraph problem. *SIAM Journal on Computing*, 22(3):573–586, 1993.

26 Dániel Marx. Parameterized graph separation problems. *Theoretical Computer Science*, 351(3):394–406, 2006.

27 Dániel Marx, Barry O'Sullivan, and Igor Razgon. Finding small separators in linear time via treewidth reduction. *ACM Transactions on Algorithms*, 9(4):30, 2013.

28 Dániel Marx and Igor Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset. *SIAM Journal on Computing*, 43(2):355–388, 2014.

29 Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proc. of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 255–264. ACM, 2008.

30 R. Ravi and Amitabh Sinha. Approximating k-cuts via network strength. In *Proc. of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 621–622. SIAM, 2002.

31 M. Stoer and F. Wagner. A simple min-cut algorithm. *Journal of the ACM*, 44(4):585–591, 1997.

32 Zoya Svitkina and Éva Tardos. Min-max multiway cut. In *Proc. of the 7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX) and 8th International Workshop on Randomization and Computation (RANDOM)*, volume 3122 of *LNCS*, pages 207–218, 2004.

Improved Exact Algorithms for Mildly Sparse Instances of Max SAT*

Takayuki Sakai[1], Kazuhisa Seto[2], Suguru Tamaki[3], and Junichi Teruyama[4]

1 **Oki Electric Industry Co., Ltd.**
2 **Seikei University, 3-3-1 Kichijoji-Kitamachi, Musashino-shi, Tokyo 180-8633, Japan**
`seto@st.seikei.ac.jp`
3 **Kyoto University, Yoshida Honmachi, Sakyo-ku, Kyoto 606-8501, Japan**
`tamak@kuis.kyoto-u.ac.jp`
4 **National Institute of Informatics, and JST, ERATO, Kawarabayashi Large Graph Project, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan**
`teruyama@nii.ac.jp`

Abstract

We present improved exponential time exact algorithms for Max SAT. Our algorithms run in time of the form $O(2^{(1-\mu(c))n})$ for instances with n variables and $m = cn$ clauses. In this setting, there are three incomparable currently best algorithms: a deterministic exponential space algorithm with $\mu(c) = \frac{1}{O(c \log c)}$ due to Dantsin and Wolpert [SAT 2006], a randomized polynomial space algorithm with $\mu(c) = \frac{1}{O(c \log^3 c)}$ and a deterministic polynomial space algorithm with $\mu(c) = \frac{1}{O(c^2 \log^2 c)}$ due to Sakai, Seto and Tamaki [Theory Comput. Syst., 2015]. Our first result is a deterministic polynomial space algorithm with $\mu(c) = \frac{1}{O(c \log c)}$ that achieves the previous best time complexity without exponential space or randomization. Furthermore, this algorithm can handle instances with exponentially large weights and hard constraints. The previous algorithms and our deterministic polynomial space algorithm run super-polynomially faster than 2^n only if $m = O(n^2)$. Our second results are deterministic exponential space algorithms for Max SAT with $\mu(c) = \frac{1}{O((c \log c)^{2/3})}$ and for Max 3-SAT with $\mu(c) = \frac{1}{O(c^{1/2})}$ that run super-polynomially faster than 2^n when $m = o(n^{5/2}/\log^{5/2} n)$ and $m = o(n^3/\log^2 n)$ respectively.

1998 ACM Subject Classification F.2.0 [Analysis of Algorithms and Problem Complexity]: General

Keywords and phrases maximum satisfiability, weighted, polynomial space, exponential space

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.90

1 Introduction

The *maximum satisfiability problem* (Max SAT) is, given a set of clauses, to find an assignment to Boolean variables that maximizes the number of satisfied clauses, where a *clause* is a disjunction of literals, a *literal* is a Boolean variable or its negation, and an assignment *satisfies* a clause if at least one literal in the clause becomes true under the assignment. Max SAT is one of the most fundamental problems in practice and theory. It is known to be

* This work was supported in part by MEXT KAKENHI (24106003); JSPS KAKENHI (25240002, 26330011, 26730007); JST ERATO Kawarabayashi Large Graph Project.

Table 1 A historical overview of upper bounds. k: an objective value, i.e., the number of constraints that must be satisfied. l: the length of an instance, i.e., the sum of arities of constraints. m: the number of constraints. n: the number of variables.

Running time	Problem	Space	Reference
$O(2^{0.4414k})$	Max SAT	polynomial	[3]
$O(2^{0.1000l})$	Max 2-SAT	polynomial	[13]
$O(2^{0.1450l})$	Max SAT	polynomial	[1]
$O(2^{0.1583m})$	Max 2-SAT	polynomial	[10]
$O(2^{0.1801m})$	Max 2-CSP	polynomial	[11]
$O(2^{0.4057m})$	Max SAT	polynomial	[5]
$O(2^{0.7909n})$	Max 2-CSP	exponential	[21, 35]
$O(2^{(1-\alpha(\frac{m}{n}))n}), \alpha(c) = \frac{1}{O(c/\log c)}$	Max 2-CSP	polynomial	[12]
$O(2^{(1-\beta(\frac{m}{n}))n}), \beta(c) = \frac{1}{O(c \log c)}$	Max SAT	exponential	[9]
$O(2^{(1-\gamma(\frac{m}{n}))n}), \gamma(c) = \frac{1}{O(2^{O(c)})}$	Max SAT	polynomial	[24]
$O(2^{(1-\delta(\frac{m}{n}))n}), \delta(c) = \frac{1}{O(c \log^3 c)}$	Max SAT	polynomial	[28] (randomized)
$O(2^{(1-\epsilon(\frac{m}{n}))n}), \epsilon(c) = \frac{1}{O(c^2 \log^2 c)}$	Max SAT	polynomial	[28] (deterministic)
$O(2^{(1-\zeta(\frac{m}{n}))n}), \zeta(c) = \frac{1}{O(c \log c)}$	Max SAT	polynomial	Theorem 1
$O(2^{(1-\eta(\frac{m}{n}))n}), \eta(c) = \frac{1}{O((c \log c)^{2/3})}$	Max SAT	exponential	Theorem 2
$O(2^{(1-\iota(\frac{m}{n}))n}), \iota(c) = \frac{1}{O(c^{1/2})}$	Max 3-SAT	exponential	Theorem 3

NP-hard. Max ℓ-SAT is a special case of Max SAT with the restriction that each clause contains at most ℓ literals. It is also NP-hard even when $\ell = 2$.

The time complexities of Max SAT and Max CSP (constraint satisfaction problem) have been studied with respect to several parameters such as an objective value k, i.e., the number of constraints that must be satisfied, the length of an instance l, i.e., the sum of arities of constraints, the number of constraints m, and the number of variables n, see, e.g., [1, 2, 3, 5, 9, 10, 11, 12, 13, 14, 16, 17, 19, 20, 21, 22, 23, 24, 26, 27, 28, 32, 33, 35]. We summarize the previous and our results in Table 1. We omit polynomial factors with respect to complexity parameters in the table. Recall that Max CSP is a generalization of Max SAT, where an instance consists of a set of arbitrary constraints instead of clauses. In Max ℓ-CSP, each constraint depends on at most ℓ variables.

An alternative way to parametrize MAX SAT is to ask whether at least $\tilde{m} + k$ clauses can be satisfied, where $\tilde{m}$ is the expected number of satisfied constraints by a uniformly random assignment, see, e.g., [15]. As for a special case of Max SAT, it is known that the satisfiability problem of CNF formulas with n variables and cn clauses can be solved in time $2^{(1-\mu(c))n}$, where $\mu(c) = 1/O(\log c)$, see [8]. As for more general problems than Max SAT, Impagliazzo, Paturi and Schneider [18] showed a satisfiability algorithm for depth-2 threshold circuits that runs in time $2^{(1-\mu(c))n}$ and exponential space for circuits with n variables and cn wires, where $\mu(c) = 1/c^{O(c^2)}$.

In this paper, we consider n and m as complexity parameters. This choice is appropriate for instances with $m = cn$ clauses, where $c > 0$ is unbounded. Given an instance with n variables and $m = cn$ clauses, Max SAT can be solved in time $\mathrm{poly}(m) \cdot 2^n$. Our goal is to design algorithms that run in time of the form $\mathrm{poly}(m) \cdot 2^{(1-\mu(c))n}$ with $\mu(c) > 0$ as large as possible. In this setting, there are three incomparable currently best algorithms: a deterministic exponential space algorithm with $\mu(c) = \frac{1}{O(c \log c)}$ due to Dantsin and Wolpert [9], a randomized polynomial space algorithm with $\mu(c) = \frac{1}{O(c \log^3 c)}$ and a deterministic polynomial space algorithm with $\mu(c) = \frac{1}{O(c^2 \log^2 c)}$ due to Sakai, Seto and Tamaki [28].

In this paper, we present an algorithm that achieves the previous best time complexity without exponential space or randomization as follows.

▶ **Theorem 1** (Polynomial Space). *Given an instance with n variables, $m = cn$ clauses and the maximum weight W, Max SAT can be solved deterministically in time* $\mathrm{poly}(m, \log W) \cdot 2^{(1-\mu(c))n}$ *and polynomial space, where* $\mu(c) = \frac{1}{O(c \log c)}$.

Furthermore, this algorithm can handle instances with hard constraints as [28]. The previous algorithms and our deterministic polynoamial space algorithm run super-polynomially faster than 2^n only if $m = O(n^2)$. In this paper, we present algorithms that run super-polynomially faster than 2^n for instances with $\omega(n^2)$ clauses with the help of exponential space as follows.

▶ **Theorem 2** (Exponential Space). *Given an instance with n variables, $m = cn$ clauses and the maximum weight W, Max SAT can be solved deterministically in time* $\mathrm{poly}(m, W) \cdot 2^{(1-\mu(c))n}$ *and exponential space, where* $\mu(c) = \frac{1}{O((c \log c)^{2/3})}$.

▶ **Theorem 3** (Exponential Space, Max 3-SAT). *Given an instance with n variables, $m = cn$ clauses and the maximum weight W, Max 3-SAT can be solved deterministically in time* $\mathrm{poly}(m, W) \cdot 2^{(1-\mu(c))n}$ *and exponential space, where* $\mu(c) = \frac{1}{O(c^{1/2})}$.

These algorithms run super-polynomially faster than 2^n for instances of Max SAT and Max 3-SAT with $m = o(n^{5/2}/\log^{5/2} n)$ and $m = o(n^3/\log^2 n)$ respectively. They can handle instances with hard constraints as well.

After this paper was submitted to this conference, the authors learned that Chen and Santhanam [7] independently obtained similar but better results than ours, i.e., a deterministic polynomial space algorithm with $\mu(c) = \frac{1}{O(c)}$ and a deterministic exponential space algorithm with $\mu(c) = \frac{1}{O(c^{1/2})}$.

1.1 Our technique

Our algorithms are based on the combination of width reduction and greedy restriction due to Sakai, Seto and Tamaki [28]. The basic idea is as follows:

1. Width reduction. Given an instance of Max SAT with n variables and m clauses, we produce a collection of instances of Max ℓ-SAT with $\ell = O(\log(m/n))$ such that the optimum of the original instance is equal to the maximum of the optima of the produced instances. Width reduction is originally invented by Schuler [31] to solve the CNF satisfiability problem, but we can modify it to handle Max SAT.

2. Greedy restriction. For each Max ℓ-SAT instance, we repeat the following until Max 1-SAT instances are obtained: Pick a variable x that appears most frequently in clauses that contain at least 2 literals, then produce two instances by setting $x = 0$ and $x = 1$. Greedy restriction is inspired by the satisfiability algorithms for Boolean formulas due to Santhanam and others [6, 30, 34].

3. Max 1-SAT. We solve Max 1-SAT instances by a polynomial time algorithm (majority voting).

In the previous analysis of greedy restriction [28], they regard each clause as a De Morgan formula and apply the so-called shrinkage lemma for De Morgan formulas due to Chen, Kabanets, Kolokolova, Shaltiel and Zuckerman [6]. In this paper, we treat each clause as

it is and improve the analysis of greedy restriction. Furthermore, if we are allowed to use exponential space, we can replace the base case of Max 1-SAT by Max 2-SAT and apply Williams' algorithm for Max 2-SAT [21, 35]. This further improves the efficiency of greedy restriction since we only have to consider clauses that contain at least 3 literals instead of at least 2 literals. Note that there is a difference between the polynomial time algorithm for Max 1-SAT and Williams' algorithm for Max 2-SAT with respect to the dependency on the maximum weight W of instances: The running time of the former and the latter involve $\mathrm{poly}(\log W)$ and $\mathrm{poly}(W)$ factors respectively as seen in Theorem 1 and Theorems 2 and 3. If $(1-\varepsilon)$-multiplicative approximation is allowed, the dependency is improved to $\mathrm{poly}(\log W)$ in Williams' algorithm, see Section 4.2 in [35].

2 Preliminaries

We denote by $\mathbb{Z}$ the set of integers. We define $-\infty$ as $-\infty + z = z + -\infty = -\infty$ and $-\infty < z$ for all $z \in \mathbb{Z}$.

Let V be a set of Boolean variables $\{x_1, \ldots, x_n\}$. We use the value 1 to indicate Boolean 'true', and 0 'false'. The *negation* of a variable $x \in V$ is denoted by $\overline{x}$. A *literal* is either a variable or its negation. An *ℓ-constraint* is a Boolean function $\phi : \{0,1\}^\ell \to \{0,1\}$, which depends on ℓ variables in V. The *width* of ϕ is ℓ. Note that a 0-constraint is either '0' or '1' (a constant function). An *ℓ-clause* is an ℓ-constraint represented as a disjunction of ℓ literals, i.e., $y_1 \vee \cdots \vee y_\ell$ for some literals $y_1, \ldots, y_\ell$.

An *instance* Φ of Max SAT consists of pairs of a constraint and a weight function, i.e., $\Phi = \{(\phi_1, w_1), \ldots, (\phi_m, w_m)\}$, where each ϕ_i is an ℓ_i-clause and $w_i : \{0,1\} \to \{-\infty\} \cup \mathbb{Z}$. We allow 0-constraints to appear in instances of Max SAT. The *width* of Φ is $\max_i \ell_i$. In Max ℓ-SAT, each instance has width at most ℓ. The *maximum weight* of Φ is $\max_{i,a:w_i(a)\neq-\infty} |w_i(a)|$. For a weight function w, we denote by $\widetilde{w}$ the weight function defined as $\widetilde{w}(1) := w(1), \widetilde{w}(0) := -\infty$. Note that a constraint with $w_i(0) = -\infty$ ($w_i(1) = -\infty$, resp.) must be satisfied (unsatisfied, resp.), i.e., it is a *hard constraint*. Without loss of generality, we do not consider instances with $w_i(0) = w_i(1) = -\infty$ for some i. We use the notation as $\mathrm{Val}(\Phi, a) := \sum_{i=1}^m w_i(\phi_i(a))$ and $\mathrm{Opt}(\Phi) := \max_{a\in\{0,1\}^n} \mathrm{Val}(\Phi, a)$.

The *length* of Φ, denoted by $L(\Phi)$, is defined as the sum of widths of clauses, i.e., $L(\Phi) := \sum_{i=1}^m \ell_i$. More generally, we define $L_k(\Phi) := \sum_{i:\ell_i \geq k} \ell_i$. We denote by $\mathrm{var}(\Phi)$ the set of variables that appear as literals in Φ. The *frequency* of a variable x with respect to Φ is the number of literals that appear in Φ as x or $\overline{x}$, and denoted by $\mathrm{freq}(\Phi, x)$. We define $\mathrm{var}_k(\Phi)$ and $\mathrm{freq}_k(\Phi, x)$ analogously to $L_k(\Phi)$.

For any instance Φ of Max SAT, any set of variables $\{x_{i_1}, \ldots, x_{i_k}\}$ and any constants $a_1, \ldots, a_k \in \{0,1\}$, we denote by $\Phi[x_{i_1} = a_1, \ldots, x_{i_k} = a_k]$ the instance obtained from Φ by assigning to each x_{i_j}, $\overline{x}_{i_j}$ the value a_j, $\overline{a}_j$ respectively and applying the following simplification rules repeatedly:

1. If Φ contains a clause of the form $\phi = 0 \vee \psi$ where ψ is a disjunction of literals, replace ϕ by the clause ψ.
2. If Φ contains a clause of the form $\phi = 1 \vee \psi$ where ψ is a disjunction of literals, replace ϕ by 1 (as a 0-constraint).

We similarly define $\phi[l_{i_1} = a_1, \ldots, l_{i_k} = a_k]$ for any set of literals $\{l_{i_1}, \ldots, l_{i_k}\}$.

3 Algorithms for Max 1-SAT and Max 2-SAT

In this section, we introduce a polynomial time algorithm for Max 1-SAT and an exponential space algorithm for Max 2-SAT. These algorithms serve as the base cases respectively in the

polynomial and exponential space algorithms for Max ℓ-SAT in the next section.

Our polynomial time algorithm for Max 1-SAT is based on simple majority voting.

▶ **Lemma 4** (Max 1-SAT). *Given an instance of Max 1-SAT* Φ ($L_2(\Phi) = 0$) *with* m *clauses and the maximum weight* W*,* $\mathrm{Opt}(\Phi)$ *can be computed in time* $\mathrm{poly}(m, \log W)$.

Proof. Let Φ be an instance $\{(\phi_1, w_1), \ldots, (\phi_m, w_m)\}$ of Max 1-SAT. Since the width of Φ is at most 1, each ϕ_i is either '0', '1', 'x_j' or '$\overline{x}_j$' (for some j). For each x_i, define $S_i(0), S_i(1)$ as

$$\begin{aligned} S_i(0) &:= \sum_{j:\phi_j = x_i} w_j(0) + \sum_{j:\phi_j = \overline{x}_i} w_j(1), \\ S_i(1) &:= \sum_{j:\phi_j = x_i} w_j(1) + \sum_{j:\phi_j = \overline{x}_i} w_j(0). \end{aligned}$$

If $S_i(0) = S_i(1)$, then set $x_i = *$ (don't care), if $S_i(0) > S_i(1)$, then set $x_i = 0$, otherwise set $x_i = 1$. This assignment achieves $\mathrm{Opt}(\Phi)$, where we assign arbitrary values to don't care variables. Operations such as addition and comparison can be done in time $\mathrm{poly}(m, \log W)$. ◀

Our exponential space algorithm for Max 2-SAT is based on Williams' algorithm for Max 2-SAT [21, 35].

▶ **Lemma 5** (Max 2-SAT). *Given an instance of Max 2-SAT* Φ ($L_3(\Phi) = 0$) *with* n *variables,* m *clauses and the maximum weight* W*,* $\mathrm{Opt}(\Phi)$ *can be computed in time* $\mathrm{poly}(m, W) \cdot 3^{n/2}$ *and exponential space.*

Proof. We use the following result.

▶ **Theorem 6** ([21, 35]). *Given an instance of Max 2-SAT with* n *variables,* m *clauses and the maximum weight* W*, where each constraint is different from the others and each weight function* w_i *satisfies* $w_i(1) > 0$ *and* $w_i(0) = 0$*,* $\mathrm{Opt}(\Phi)$ *can be computed in time* $\mathrm{poly}(m, W) \cdot 2^{\omega n/3}$ *and exponential space, where* $\omega < 2.3728639$ [25] denotes the exponent of the matrix multiplication.

Before applying Williams' algorithm, we need preprocessing on a given instance because our instances are more general than those considered in [21, 35]. That is, we consider instances that contain two or more identical clauses, arbitrary weighted functions and hard constraints. We transform instances to a set of instances that do not contain identical clauses and whose weight functions are of the form $w_i(1) > 0$ and $w_i(0) = 0$. We ensure that the optimum of the original instance is equal to the maximum of the optima of the resulting instances. The preprocessing increases the running time, i.e., $3^{n/2} > 2^{\omega n/3}$.

Let Φ be an instance $\{(\phi_1, w_1), \ldots, (\phi_m, w_m)\}$ of Max 2-SAT. For simplicity, we treat only instances with $w_i(1) \geq w_i(0)$ for all i. The restriction on instances can be removed with additional transformation.

First, we replace each $w_i(a)$ with $w_i(0) \neq -\infty$ by $w_i(a) - w_i(0)$ to satisfy $w_i(1) > 0$ and $w_i(0) = 0$. This transformation increases the maximum weight of the instance by at most twice. We define $W_0 := \sum_{i:w_i(0) \neq -\infty} w_i(0)$ and use this later to reset the offset.

Next, we reduce the number of clauses that appear twice or more in the instance as follows: If the instance contains (ϕ_i, w_i) and (ϕ_j, w_j) such that $\phi_i = \phi_j$, then replace (ϕ_i, w_i) by $(\phi_i, w_i + w_j)$ and remove (ϕ_j, w_j). The maximum weight of the instance becomes at most $O(mW)$.

Finally, we remove hard constraints and apply Williams' algorithm. To do so, we consider a *maximal independent set* of hard constraints constructed as follows: Set $I = \emptyset$ and repeat the following: Pick arbitrary (ϕ_i, w_i) in Φ such that $w_i(0) = -\infty$ and add it to I if ϕ_i is *independent* of I, i.e., the literals in ϕ_i do not appear in any clause in I. When I becomes a maximal independent set, the cardinality of I, denoted by $|I|$, is at most $n/2$.

We try every assignment that satisfies all the clauses in I one by one. The number of such assignments is $3^{|I|}$. Note that if we assign values to all the literals that appear in some clause in I, then we obtain an instance whose hard constraints are 0 or 1-constraints. Since hard 1-constraints only determine the values of some variables and hard 0-constraints determine the feasibility of the instance, now we can apply Williams' algorithm. Note that we must add W_0 to the result of Williams' algorithm to obtain $\mathrm{Opt}(\Phi)$. The overall running time is $\mathrm{poly}(m, W) \cdot 3^{|I|} \cdot 2^{(\omega/3)(n-2|I|)} \leq \mathrm{poly}(m, W) \cdot 3^{n/2}$. ◀

4 Greedy Restriction Algorithms for Max ℓ-SAT

In this section, we present polynomial and exponential space algorithms for Max ℓ-SAT based on the combination of greedy restriction and the algorithms in the previous section. These serve as subroutines in our main algorithms for Max SAT. For Max 3-SAT, our exponential space algorithm for Max ℓ-SAT runs in time as stated in Theorem 3. First, we introduce shrinkage lemmas that are useful in the analysis of greedy restriction. Then, we describe our algorithms and their analyses.

4.1 Shrinkage Lemmas

In this section, we provide shrinkage lemmas that are useful in upper-bounding the length of instances with respect to $L_k(\cdot)$ after a sequence of greedy restriction. For an instance Φ of Max SAT with n variables, we define a sequence of random variables $\Phi_0, \Phi_1, \ldots, \Phi_n$ inductively as follows: $\Phi_0 := \Phi$. Given $\Phi_0, \Phi_1, \ldots, \Phi_{i-1}$, $\Phi_i := \Phi_{i-1}[x = a]$, where $x = \arg\max_{x \in \mathrm{var}(\Phi_{i-1})} \mathrm{freq}_k(\Phi_{i-1}, x)$ and a is a uniform random bit. We have the following lemma.

▶ **Lemma 7** (Shrinkage of Max SAT instances with respect to $L_k(\cdot)$, Lemma 4.2 in [29]). *For $n' \geq 4$, we have*

$$\Pr\left[L_k(\Phi_{n-n'}) \geq 2^k \cdot L_k(\Phi) \cdot \left(\frac{n'}{n}\right)^{\frac{k+2}{2}}\right] < 2^{-n'}.$$

Note that the shrinkage lemma used in [28] is only for $k = 2$ and provides the bound

$$\Pr\left[L_2(\Phi_{n-n'}) \geq 2 \cdot L_2(\Phi) \cdot \left(\frac{n'}{n}\right)^{\frac{3}{2}}\right] < 2^{-n'}.$$

Thus, Lemma 7 generalizes the above by introducing a parameter k and also improves the bound of it. For instances of Max 3-SAT with n variables and $k = 3$, we further improve the bound of Lemma 7 as follows.

▶ **Lemma 8** (Shrinkage of Max 3-SAT instances with respect to $L_3(\cdot)$). *For $n' \geq 1$, we have*

$$L_3(\Phi_{n-n'}) \leq L_3(\Phi) \cdot \left(\frac{n'}{n}\right)^3.$$

MaxℓSAT($\Phi = \{(\phi_1, w_1), \ldots, (\phi_m, w_m)\}$: instance, n, n': integer)
01: **if** $n > n'$, **then**
02: $x \leftarrow \arg\max_{x \in \mathrm{var}(\Phi)} \mathrm{freq}_2(\Phi, x)$.
03: $K_0 \leftarrow$ **MaxℓSAT**$(\Phi[x = 0], n - 1, n')$.
04: $K_1 \leftarrow$ **MaxℓSAT**$(\Phi[x = 1], n - 1, n')$.
05: **return** $\max\{K_0, K_1\}$.
06: **else**
07: **return** $\mathrm{Opt}(\Phi)$ by Lemma 9.

Figure 1 Algorithm for Max ℓ-SAT.

Proof. Let $x = \arg\max_{x \in \mathrm{var}(\Phi_i)} \mathrm{freq}_3(\Phi_i, x)$. For all $a \in \{0, 1\}$, we have $L_3(\Phi_i[x = a]) = L_3(\Phi_i) - 3 \cdot \mathrm{freq}_3(\Phi_i, x)$ because if a clause of width 3 contains x as a literal, it becomes a clause of width either 0 or 2 by assigning a to x. Since $\mathrm{freq}_3(\Phi_i, x) \geq \frac{L_3(\Phi_i)}{n-i}$, for all $a \in \{0, 1\}$, we have

$$L_3(\Phi_i[x = a]) = L_3(\Phi_i) - 3 \cdot \mathrm{freq}(\Phi_i, x) \leq \left(1 - \frac{3}{n-i}\right) \cdot L_3(\Phi_i) \leq L_3(\Phi_i) \cdot \left(1 - \frac{1}{n-i}\right)^3.$$

We complete the proof by induction. ◀

4.2 A Polynomial Space Algorithm

Before presenting our polynomial space algorithm for Max ℓ-SAT, we show a simple algorithm for Max SAT that is efficient for instances with a small number of clauses of width at least 2.

▶ **Lemma 9** (Algorithm for "almost" Max 1-SAT instances). *Given an instance Φ of Max ℓ-SAT with m clauses and the maximum weight W, $\mathrm{Opt}(\Phi)$ can be computed in time $\mathrm{poly}(m, \log W) \cdot 2^{L_2(\Phi)}$.*

Proof. Recall that $\mathrm{var}_2(\Phi)$ is the set of variables that appear in clauses of width at least two. We assume $\mathrm{var}_2(\Phi) = \{x_{i_1}, x_{i_2}, \ldots, x_{i_{|\mathrm{var}_2(\Phi)|}}\}$. For each assignment $a_1, a_2, \ldots, a_{|\mathrm{var}_2(\Phi)|} \in \{0, 1\}$ to $\mathrm{var}_2(\Phi)$, we can compute $\mathrm{Opt}(\Phi[x_{i_1} = a_1, x_{i_2} = a_2, \ldots, x_{i_{|\mathrm{var}_2(\Phi)|}} = a_{|\mathrm{var}_2(\Phi)|}])$ in time $\mathrm{poly}(m, \log W)$ by Lemma 4 since $L_2(\Phi[x_{i_1} = a_1, x_{i_2} = a_2, \ldots, x_{i_{|\mathrm{var}_2(\Phi)|}} = a_{|\mathrm{var}_2(\Phi)|}]) = 0$. Define

$$K := \max_{a_1, a_2, \ldots, a_{|\mathrm{var}_2(\Phi)|} \in \{0,1\}} \mathrm{Opt}(\Phi[x_{i_1} = a_1, x_{i_2} = a_2, \ldots, x_{i_{|\mathrm{var}_2(\Phi)|}} = a_{|\mathrm{var}_2(\Phi)|}]),$$

then $\mathrm{Opt}(\Phi) = K$ holds. The value K can be obtained in time $\mathrm{poly}(m, \log W) \cdot 2^{L_2(\Phi)}$ since $|\mathrm{var}_2(\Phi)| \leq L_2(\Phi)$. ◀

Our polynomial space algorithm for Max ℓ-SAT is shown in Fig. 1. The same algorithm and its analysis for the so-called Max De Morgan formula SAT was given by [28]. We analyze the running time of the algorithm only for instances of Max ℓ-SAT, which is a special case of Max De Morgan formula SAT, and obtain a better upper bound than that of [28]. Note that we may break ties arbitrarily in arg max in Line 02. In particular, when $\mathrm{var}_2(\Phi) = \emptyset$, we may choose any variable x. In what follows, we give the running time analysis of the algorithm.

▶ **Lemma 10** (Polynomial Space Algorithm for Max ℓ-SAT). *Given an instance Φ of Max ℓ-SAT with n variables, $m = cn$ clauses and the maximum weight W, $\mathrm{Opt}(\Phi)$ can be computed in time $\mathrm{poly}(m, \log W) \cdot 2^{\left(1 - \frac{1}{16c\ell}\right)n}$ and polynomial space.*

Proof. Let Φ be an instance $\{(\phi_1, w_1), \ldots, (\phi_m, w_m)\}$ of Max ℓ-SAT with n variables, $m = cn$ clauses and the maximum weight W. Note that $L_2(\Phi) \leq \ell m = c\ell n$. We set the parameter n' in the algorithm as $n' = \frac{n}{8c\ell}$. For simplicity, we assume n' is an integer.

Let us regard the computation of the algorithm as a complete binary tree of depth $n - n'$ inductively defined as follows. The root is labeled with Φ. The left and the right children of the root are labeled with $\Phi[x = 0]$ and $\Phi[x = 1]$ respectively, where $x = \arg\max_{x \in \mathrm{var}(\Phi)} \mathrm{freq}_2(\Phi, x)$. We define the children of $\Phi[x = 0]$ and $\Phi[x = 1]$ in the similar way and so on until depth $n - n'$ is reached.

To upper-bound the running time of the algorithm, we are interested in $L_2(\cdot)$ of instances that appear as leaves. Pick any leaf and assume its label is Ψ. Then, the algorithm executes Lines 06–07 on Ψ and it takes $\mathrm{poly}(m, \log W) \cdot 2^{L_2(\Phi)}$ time.

To estimate the average of $L_2(\Psi)$, we can apply Lemma 7 because the random sequence $\{\Phi_i\}$ is exactly associated with the complete binary tree defined above. In particular, if we pick a leaf of the tree uniformly at random, then the distribution of its label is exactly the same as that of $\Phi_{n-n'}$.

By Lemma 7 with $k = 2$, we can bound the fraction of leaves labeled with Ψ, $L_2(\Psi) \geq n'/2$, from above by $2^{-n'}$ due to the choice of n'. Thus, the overall running time of the algorithm is at most

$$\mathrm{poly}(m, \log W) \cdot 2^{n-n'} \cdot \{2^{-n'} \cdot 2^{n'} + (1 - 2^{-n'}) \cdot 2^{n'/2}\} = \mathrm{poly}(m, \log W) \cdot 2^{\left(1 - \frac{1}{16c\ell}\right)n}. \quad \blacktriangleleft$$

4.3 An Exponential Space Algorithm

Our exponential space algorithm for Max ℓ-SAT is almost the same as our polynomial space algorithm except that we use $L_3(\cdot)$ and the following lemma instead of $L_2(\cdot)$ and Lemma 9.

▶ **Lemma 11** (Algorithm for "almost" Max 2-SAT instances). *Given an instance Φ of Max ℓ-SAT with n variables, m clauses and the maximum weight W,* $\mathrm{Opt}(\Phi)$ *can be computed in time* $\mathrm{poly}(m, W) \cdot 2^{L_3(\Phi)} \cdot 3^{(n - L_3(\Phi))/2}$ *and exponential space.*

Proof. Assume $\mathrm{var}_3(\Phi) = \{x_{i_1}, x_{i_2}, \ldots, x_{i_{|\mathrm{var}_3(\Phi)|}}\}$. For each assignment $a_1, a_2, \ldots, a_{|\mathrm{var}_3(\Phi)|} \in \{0, 1\}$ to $\mathrm{var}_3(\Phi)$, we can compute $\mathrm{Opt}(\Phi[x_{i_1} = a_1, x_{i_2} = a_2, \ldots, x_{i_{|\mathrm{var}_3(\Phi)|}} = a_{|\mathrm{var}_3(\Phi)|}])$ in time $\mathrm{poly}(m, W) \cdot 3^{(n - |\mathrm{var}_3(\Phi)|)/2}$ and exponentiail space by Lemma 5 since $L_3(\Phi[x_{i_1} = a_1, x_{i_2} = a_2, \ldots, x_{i_{|\mathrm{var}_3(\Phi)|}} = a_{|\mathrm{var}_3(\Phi)|}]) = 0$. Define

$$K := \max_{a_1, a_2, \ldots, a_{|\mathrm{var}_3(\Phi)|} \in \{0,1\}} \mathrm{Opt}(\Phi[x_{i_1} = a_1, x_{i_2} = a_2, \ldots, x_{i_{|\mathrm{var}_3(\Phi)|}} = a_{|\mathrm{var}_3(\Phi)|}]),$$

then $\mathrm{Opt}(\Phi) = K$ holds. The value K can be obtained in time $\mathrm{poly}(m, W) \cdot 2^{L_3(\Phi)} \cdot 3^{(n - L_3(\Phi))/2}$ and exponential space since $|\mathrm{var}_3(\Phi)| \leq L_3(\Phi)$. $\blacktriangleleft$

We modify **MaxℓSAT** in Fig. 1 as follows: (1) In Line 02, replace freq_2 by freq_3, (2) in Line 07, replace Lemma 9 by Lemma 11. The resulting algorithm yields the following.

▶ **Lemma 12** (Exponential Space Algorithm for Max ℓ-SAT). *Given an instance of Max ℓ-SAT with n variables, $m = cn$ constrains and the maximum weight W,* $\mathrm{Opt}(\Phi)$ *can be computed in time* $\mathrm{poly}(m, W) \cdot 2^{(1 - \mu(c))n}$ *and exponential space, where* $\mu(c) = \frac{1}{O((c\ell)^{2/3})}$.

▶ **Theorem 13** (Restatement of Theorem 3). *Given an instance with n variables, $m = cn$ clauses and the maximum weight W, Max 3-SAT can be solved deterministically in time* $\mathrm{poly}(m, W) \cdot 2^{(1 - \mu(c))n}$ *and exponential space, where* $\mu(c) = \frac{1}{O(c^{1/2})}$.

The proofs of the above lemma and theorem are almost as the same as that of Lemma 10 except that we apply Lemma 7 with $k = 3, n' = \frac{n}{(16c\ell)^{2/3}}$ and Lemma 8 with $n' = \frac{n}{\sqrt{6c}}$ respectively instead of Lemma 7 with $k = 2, n' = \frac{n}{8c\ell}$.

MaxSAT($\Phi = \{(\phi_1, w_1), \ldots, (\phi_m, w_m)\}$: **instance,** n, ℓ: **integer**)
01: **if** $L_{\ell+1}(\Phi) = 0$, **then**
02: **return Max**ℓ**SAT**(Φ, n, n'). /* $n' = \frac{n}{8(m/n)\ell}$ */
03: **else**
04: Pick arbitrary $\phi_i = (l_1 \vee \cdots \vee l_{\ell'})$ such that $\ell' > \ell$.
05: $\Phi_L \leftarrow \{\Phi \setminus \{(\phi_i, w_i)\}\} \cup \{(l_1 \vee \cdots \vee l_\ell, \widetilde{w_i})\}$.
06: $K_L \leftarrow$ **MaxSAT**(Φ_L, n, ℓ).
07: $\Phi_R \leftarrow \Phi[l_1 = \cdots = l_\ell = 0]$.
08: $K_R \leftarrow$ **MaxSAT**$(\Phi_R, n - \ell, \ell)$.
09: return $\max\{K_L, K_R\}$.

Figure 2 Max SAT algorithm.

5 Width Reduction Algorithms for Max SAT

In this section, we present polynomial and exponential space algorithms for Max SAT based on the combination of width reduction and the algorithms in the previous section, completing the proof of Theorems 1 and 2.

Our polynomial space algorithm for Max SAT is shown in Fig. 2. Again, the same algorithm and its analysis was given by [28] but we obtain a better upper bound on the running time because we use a faster Max ℓ-SAT algorithm than that of [28] as a subroutine. In what follows, we show the running time analysis of the algorithm.

▶ **Theorem 14** (Restatement of Theorem 1). *Given an instance with n variables, $m = cn$ clauses and the maximum weight W, Max SAT can be solved deterministically in time* $\mathrm{poly}(m, \log W) \cdot 2^{(1-\mu(c))n}$ *and polynomial space, where* $\mu(c) = \frac{1}{O(c \log c)}$.

Proof. The overall structure of the proof is similar to the analysis of width reduction for the CNF satisfiability problem due to Calabro et al. [4]. We think of the execution of **MaxSAT** as a rooted binary tree T, i.e., the root of T is labeled with an input instance Φ and for each node labeled with Ψ, its left (right, resp.) child is labeled with Ψ_L (Ψ_R, resp.) as defined in Line 05 (Line 07, resp.) in the algorithm. If Ψ is an instance of Max ℓ-SAT, i.e., every clause ψ_i in Ψ has width at most ℓ, then the node labeled with Ψ is a leaf.

Let us consider a path p from the root to a leaf v labeled with Ψ. We denote by L and R the number of left and right children p selects to reach v. It is easy to see that (1) $L \leq m$ since the number of clauses is m, (2) $R \leq n/\ell$ since a right branch eliminates ℓ variables at a time, and (3) Ψ is defined over at most $n - R\ell$ variables. Furthermore, the number of leaves which are reachable by exactly R times of right branches is at most $\binom{m+R}{R}$. Let $T(n, m, W, \ell)$ denote the running time of **Max**ℓ**SAT** on instances of Max ℓ-SAT with n variables, $m = cn$ clauses and the maximum weight W due to Lemma 10. We can upper bound the running time of **MaxSAT** as:

$$\mathrm{poly}(m, \log W) \left(\sum_{R=0}^{\frac{n}{2\ell}-1} \binom{m+R}{R} T(n - R\ell, m, W, \ell) + \sum_{R=\frac{n}{2\ell}}^{\frac{n}{\ell}} \binom{m+R}{R} T(n - R\ell, m, W, \ell) \right).$$

In what follows, we omit $\mathrm{poly}(m, \log W)$ factors due to space limitations. That is, the following inequalities hold if we ignore $\mathrm{poly}(m, \log W)$ factors. We first upper bound the second summation above.

$$\left(\sum_{R=\frac{n}{2\ell}}^{\frac{n}{\ell}}\binom{m+R}{R}T(n-R\ell,m,W,\ell)\right)\leq\left(\sum_{R=\frac{n}{2\ell}}^{\frac{n}{\ell}}\binom{m+R}{R}2^{n-R\ell}\right)$$
$$\leq\binom{m+\frac{n}{2\ell}}{\frac{n}{2\ell}}2^{n/2}\leq\binom{2m}{\frac{n}{2\ell}}2^{n/2}\leq 2^{\frac{n\log(4c\ell)}{\ell}}\cdot 2^{n/2}\leq 2^{(1-\mu(c))n},$$

where we set $\ell=\alpha\log c$ for sufficiently large constant $\alpha>0$ and the last inequality follows from $\binom{n}{\beta n}\leq\mathrm{poly}(n)\cdot 2^{2\beta n\log(1/\beta)}$ for $\beta\leq 1/2$.

We move on to the analysis of the first summation.

$$\left(\sum_{R=0}^{\frac{n}{2\ell}-1}\binom{m+R}{R}T(n-R\ell,m,W,\ell)\right)\leq\left(\sum_{R=0}^{\frac{n}{2\ell}-1}\binom{m+R}{R}2^{\left(1-\frac{1}{16\ell(\frac{cn}{n-R\ell})}\right)(n-R\ell)}\right)$$
$$\leq\left(\sum_{R=0}^{m+\frac{n}{2\ell}}\binom{m+\frac{n}{2\ell}}{R}2^{\left(1-\frac{1}{32\ell c}\right)(n-R\ell)}\right)=\left(2^{\left(1-\frac{1}{32\ell c}\right)n}\left(1+2^{-\left(1-\frac{1}{32\ell c}\right)\ell}\right)^{m+\frac{n}{2\ell}}\right)$$
$$\leq\left(2^{\left(1-\frac{1}{32\ell c}\right)n}\left(e^{2^{-\left(1-\frac{1}{32\ell c}\right)\ell}}\right)^{m+\frac{n}{2\ell}}\right)\leq\left(2^{\left(1-\frac{1}{32\ell c}\right)n+\frac{2m}{2^{\left(1-\frac{1}{32\ell c}\right)\ell}}}\right).$$

Since we set $\ell=\alpha\log c$ for sufficiently large constant $\alpha>0$, the exponent is

$$\left(1-\frac{1}{32\ell c}\right)n+\frac{2m}{2^{\left(1-\frac{1}{32\ell c}\right)\ell}}=\left(1-\frac{1}{32\alpha c\log c}\right)n+\frac{2cn}{2^{\alpha\log c-\frac{1}{32c}}}\leq\left(1-\frac{1}{64\alpha c\log c}\right)n,$$

where the last inequality is by the choice of α and $c>4$. This completes the proof. ◀

If we use the modified algorithm for Max ℓ-SAT due to Lemma 12 in Line 02 in Fig. 2 with $n'=\frac{n}{(16c\ell)^{2/3}}$, we have:

▶ **Theorem 15** (Restatement of Theorem 2). *Given an instance with n variables, $m=cn$ clauses and the maximum weight W, Max SAT can be solved deterministically in time $\mathrm{poly}(m,W)\cdot 2^{(1-\mu(c))n}$ and exponential space, where $\mu(c)=\frac{1}{O((c\log c)^{2/3})}$.*

The proof of the above theorem is almost identical to that of Theorem 14 and we omit it.

6 Concluding Remarks

There are several open questions. First, can we show a (randomized) polynomial space algorithm that runs super-polynomially faster than 2^n for instances with $\omega(n^2)$ clauses? One possible way is to design a non-trivial polynomial space algorithm for instances of Max 2-SAT with arbitrary number of clauses. Second, can we modify our exponential space algorithms to handle instances with exponentially large weights? To do so, we might have to show a non-trivial algorithm for instances of Max 2-SAT with arbitrary number of clauses and exponentially large weights.

Finally, we remark that the recent work of the authors [29] shows a deterministic $2^{n-n^{1/O(\ell)}}$ time and exponential space algorithm for instances of Max SAT with $m=O(n^\ell)$ clauses.

Acknowledgements. We are grateful to Osamu Watanabe, who asked us whether [28] can be improved if we treat each constraint as a clause instead of a De Morgan formula. Without his question, this work would not exist. We would like to thank anonymous reviewers for their helpful comments on our paper.

References

1 Nikhil Bansal and Venkatesh Raman. Upper bounds for MaxSAT: Further improved. In *Proceedings of the 10th International Symposium on Algorithms and Computation (ISAAC)*, volume 1741 of *Lecture Notes in Computer Science*, pages 247–258. Springer, 1999.

2 Daniel Binkele-Raible and Henning Fernau. A new upper bound for Max-2-SAT: A graph-theoretic approach. *J. Discrete Algorithms*, 8(4):388–401, 2010.

3 Ivan Bliznets and Alexander Golovnev. A new algorithm for parameterized MAX-SAT. In *Proceedings of the 7th International Symposium on Parameterized and Exact Computation (IPEC)*, volume 7535 of *Lecture Notes in Computer Science*, pages 37–48. Springer, 2012.

4 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. A duality between clause width and clause density for SAT. In *Proceedings of the 21st Annual IEEE Conference on Computational Complexity (CCC)*, pages 252–260, 2006.

5 Jianer Chen and Iyad A. Kanj. Improved exact algorithms for MAX-SAT. *Discrete Applied Mathematics*, 142(1-3):17–27, 2004.

6 Ruiwen Chen, Valentine Kabanets, Antonina Kolokolova, Ronen Shaltiel, and David Zuckerman. Mining circuit lower bound proofs for meta-algorithms. *Computational Complexity*, 24(2):333–392, 2015.

7 Ruiwen Chen and Rahul Santhanam. Improved algorithms for sparse max-sat and max-k-csp. In *Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 2015. To appear.

8 Evgeny Dantsin and Edward A. Hirsch. Worst-case upper bounds. In *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 403–424. IOS Press, 2009.

9 Evgeny Dantsin and Alexander Wolpert. MAX-SAT for formulas with constant clause density can be solved faster than in $O(2^n)$ time. In *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 4121 of *Lecture Notes in Computer Science*, pages 266–276. Springer, 2006.

10 Serge Gaspers and Gregory B. Sorkin. A universally fastest algorithm for Max 2-SAT, Max 2-CSP, and everything in between. *J. Comput. Syst. Sci.*, 78(1):305–335, 2012.

11 Serge Gaspers and Gregory B. Sorkin. Separate, measure and conquer: Faster polynomial-space algorithms for Max 2-CSP and counting dominating sets. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP), Part I*, pages 567–579, 2015.

12 Alexander Golovnev and Konstantin Kutzkov. New exact algorithms for the 2-constraint satisfaction problem. *Theor. Comput. Sci.*, 526:18–27, 2014.

13 Jens Gramm, Edward A. Hirsch, Rolf Niedermeier, and Peter Rossmanith. Worst-case upper bounds for MAX-2-SAT with an application to MAX-CUT. *Discrete Applied Mathematics*, 130(2):139–155, 2003.

14 Jens Gramm and Rolf Niedermeier. Faster exact solutions for MAX2SAT. In *Proceedings of the 4th Italian Conference on Algorithms and Complexity (CIAC)*, volume 1767 of *Lecture Notes in Computer Science*, pages 174–186. Springer, 2000.

15 Gregory Gutin and Anders Yeo. Constraint satisfaction problems parameterized above or below tight bounds: A survey. In *The Multivariate Algorithmic Revolution and Beyond*, volume 7370 of *Lecture Notes in Computer Science*, pages 257–286. Springer, 2012.

16 Edward A. Hirsch. A new algorithm for MAX-2-SAT. In *Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 1770 of *Lecture Notes in Computer Science*, pages 65–73. Springer, 2000.

17 Edward A. Hirsch. Worst-case study of local search for MAX-k-SAT. *Discrete Applied Mathematics*, 130(2):173–184, 2003.

18 Russell Impagliazzo, Ramamohan Paturi, and Stefan Schneider. A satisfiability algorithm for sparse depth two threshold circuits. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 479–488, 2013.

19 Joachim Kneis, Daniel Mölle, Stefan Richter, and Peter Rossmanith. On the parameterized complexity of exact satisfiability problems. In *Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 3618 of *Lecture Notes in Computer Science*, pages 568–579. Springer, 2005.

20 Joachim Kneis, Daniel Mölle, Stefan Richter, and Peter Rossmanith. A bound on the pathwidth of sparse graphs with applications to exact algorithms. *SIAM J. Discrete Math.*, 23(1):407–427, 2009.

21 Mikko Koivisto. Optimal 2-constraint satisfaction via sum-product algorithms. *Inf. Process. Lett.*, 98(1):24–28, 2006.

22 Arist Kojevnikov and Alexander S. Kulikov. A new approach to proving upper bounds for MAX-2-SAT. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 11–17, 2006.

23 Alexander S. Kulikov. Automated generation of simplification rules for SAT and MAXSAT. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 3569 of *Lecture Notes in Computer Science*, pages 430–436. Springer, 2005.

24 Alexander S. Kulikov and Konstantin Kutzkov. New bounds for MAX-SAT by clause learning. In *Proceedings of the 8th International Computer Science Symposium in Russia (CSR)*, volume 4649 of *Lecture Notes in Computer Science*, pages 194–204. Springer, 2007.

25 François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 296–303, 2014.

26 Meena Mahajan and Venkatesh Raman. Parameterizing above guaranteed values: MaxSAT and MaxCUT. *J. Algorithms*, 31(2):335–354, 1999.

27 Rolf Niedermeier and Peter Rossmanith. New upper bounds for maximum satisfiability. *J. Algorithms*, 36(1):63–88, 2000.

28 Takayuki Sakai, Kazuhisa Seto, and Suguru Tamaki. Solving sparse instances of max SAT via width reduction and greedy restriction. *Theory Comput. Syst.*, 57(2):426–443, 2015.

29 Takayuki Sakai, Kazuhisa Seto, Suguru Tamaki, and Junichi Teruyama. A satisfiability algorithm for depth-2 circuits with a symmetric gate at the top and AND gates at the bottom. *Electronic Colloquium on Computational Complexity (ECCC)*, TR15-136, 2015.

30 Rahul Santhanam. Fighting perebor: New and improved algorithms for formula and QBF satisfiability. In *Proceedings of the 51th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 183–192, 2010.

31 Rainer Schuler. An algorithm for the satisfiability problem of formulas in conjunctive normal form. *J. Algorithms*, 54(1):40–44, 2005.

32 Alex D. Scott and Gregory B. Sorkin. Faster algorithms for MAX CUT and MAX CSP, with polynomial expected time for sparse instances. In *Proceedings of the 6th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX) and the 7th International Workshop on Randomization and Computation (RANDOM)*, volume 2764 of *Lecture Notes in Computer Science*, pages 382–395. Springer, 2003.

33 Alexander D. Scott and Gregory B. Sorkin. Linear-programming design and analysis of fast algorithms for Max 2-CSP. *Discrete Optimization*, 4(3-4):260–287, 2007.

34 Kazuhisa Seto and Suguru Tamaki. A satisfiability algorithm and average-case hardness for formulas over the full binary basis. *Computational Complexity*, 22(2):245–274, 2013.

35 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005.

Polynomial Fixed-parameter Algorithms: A Case Study for Longest Path on Interval Graphs*

Archontia C. Giannopoulou†[1], George B. Mertzios[2], and Rolf Niedermeier[3]

1 **Institute of Informatics, University of Warsaw, Poland**
archontia.giannopoulou@gmail.com
2 **School of Engineering and Computing Sciences, Durham University, UK**
george.mertzios@durham.ac.uk
3 **Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany**
rolf.niedermeier@tu-berlin.de

Abstract

We study the design of fixed-parameter algorithms for problems already known to be solvable in polynomial time. The main motivation is to get more efficient algorithms for problems with unattractive polynomial running times. Here, we focus on a fundamental graph problem: Longest Path; it is NP-hard in general but known to be solvable in $O(n^4)$ time on n-vertex interval graphs. We show how to solve Longest Path on Interval Graphs, parameterized by vertex deletion number k to proper interval graphs, in $O(k^9 n)$ time. Notably, Longest Path is trivially solvable in linear time on proper interval graphs, and the parameter value k can be approximated up to a factor of 4 in linear time. From a more general perspective, we believe that using parameterized complexity analysis for polynomial-time solvable problems offers a very fertile ground for future studies for all sorts of algorithmic problems. It may enable a refined understanding of efficiency aspects for polynomial-time solvable problems, similarly to what classical parameterized complexity analysis does for NP-hard problems.

1998 ACM Subject Classification G.2.2 Graph Theory – Graph Algorithms, Path and Circuit Problems, F.2.2 Nonnumerical Algorithms and Problems – Computations on Discrete Structures

Keywords and phrases fixed-parameter algorithm, preprocessing, data reduction, polynomial-time algorithm, longest path problem, interval graphs, proper interval vertex deletion set

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.102

1 Introduction

Parameterized complexity analysis [16, 18, 30] is a flourishing field dealing with the exact solvability of NP-hard problems. The key idea is to lift classical complexity analysis, rooted in the P versus NP phenomenon, from a one-dimensional to a two- (or even multi-)dimensional perspective, the key concept being "fixed-parameter tractability (FPT)". But why should this natural and successful approach be limited to intractable (i.e., NP-hard) problems? We are convinced that appropriately parameterizing polynomially solvable problems sheds new light on what makes a problem far from being solvable in *linear* time, in the same way as

* Partially supported by the EPSRC Grant EP/K022660/1 and the Warsaw Center of Mathematics and Computer Science.
† The main part of this paper was prepared while the author was affiliated at the School of Engineering and Computing Sciences, Durham University, UK.

classical FPT algorithms help in illuminating what makes an NP-hard problem far from being solvable in *polynomial* time. In a nutshell, the credo and leitmotif of this paper is that "FPT inside P" is a very interesting, but still too little explored, line of research.

The known results fitting under this leitmotif are somewhat scattered around in the literature and do not systematically refer to or exploit the toolbox of parameterized algorithm design. This should change and "FPT inside P" should be placed on a much wider footing, using parameterized algorithm design techniques such as data reduction and kernelization. As a simple illustrative example, consider the Maximum Matching problem. By following a "Buss-like" kernelization (as is standard knowledge in parameterized algorithmics [16, 30]) and then applying a known polynomial-time matching algorithm, it is not difficult to derive an efficient algorithm that, given a graph G with n vertices, computes a matching of size at least k in $O(kn + k^3)$ time.

More formally, and somewhat more generally, we propose the following scenario. Given a problem with instance size n for which there exists an $O(n^c)$-time algorithm, our aim is to identify appropriate parameters k and to derive algorithms with time complexity $f(k) \cdot n^{c'}$ such that $c' < c$, where $f(k)$ depends only on k. First we refine the class FPT by defining, for every polynomially-bounded function $p(n)$, the class FPT($p(n)$) containing the problems solvable in $f(k) \cdot p(n)$ time, where $f(k)$ is an arbitrary (possibly exponential) function of k. It is important to note that, in strong contrast to FPT algorithms for NP-hard problems, here the function $f(k)$ may also become *polynomial* in k. Motivated by this, we refine the class P by introducing, for every polynomial $p(n)$, the class *P-FPT*($p(n)$) *(Polynomial Fixed-Parameter Tractable)*, containing the problems solvable in $O(k^t \cdot p(n))$ time for some constant $t \geq 1$, i.e., the dependency of the complexity on the parameter k is at most polynomial. In this paper we focus our attention on the (practically perhaps most attractive) subclass *PL-FPT (Polynomial-Linear Fixed-Parameter Tractable)*, where PL-FPT = P-FPT(n). For example, the algorithm we sketched above for Maximum Matching, parameterized by solution size k, belongs to PL-FPT.

In an attempt to systematically follow the leitmotif "FPT inside P", we put forward three desirable algorithmic properties:

1. The running time should have a polynomial dependency on the parameter.
2. The running time should be as close to linear as possible if the parameter value is constant, improving upon an existing "high-degree" polynomial-time (unparameterized) algorithm.
3. The parameter value, or a good approximation thereof, should be computable efficiently (preferably in linear time) for arbitrary parameter values.

In addition, as this research direction is still only little explored, we suggest and follow a focus first on problems for which the best known upper bounds of the time complexity are polynomials of high degree, e.g., $O(n^4)$ or higher.

Related work

Here we discuss previous work on graph problems that fits under the leitmotif "FPT inside P"; however there exists further related work also in other topics such as string matching [6] or Linear Program solving [28].

The complexity of some known polynomial-time algorithms can be easily "tuned" with respect to specific parameters, thus immediately reducing the complexity whenever these parameters are bounded. For instance, in n-vertex and m-edge graphs with nonnegative edge weights, Dijkstra's $O(m+n \log n)$-time algorithm for computing shortest paths can be adapted to an $O(m + n \log k)$-time algorithm, where k is the number of distinct edge weights [27] (also refer to [31]). In addition, motivated by the quest for explaining the efficiency of several

shortest path heuristics for road networks (where Dijkstra's algorithm is too slow for routing applications), the "highway dimension" was introduced [4] as a parameterization helping to do rigorous proofs about the quality of the heuristics. Altogether, the work on shortest path computations shows that even for quasi-linear-time algorithms adopting a parameterized view may be of significant practical interest.

Maximum flow computations constitute another important application area for "FPT inside P". An $O(k^3 n \log n)$-time maximum flow algorithm was presented [22] for graphs that can be made planar by deleting k "crossing edges"; notably, here it is assumed that the embedding and the k crossing edges are given along with the input. An $O(g^8 n \log^2 n \log^2 C)$-time maximum flow algorithm was developed [12], where g is the genus of the graph and C is the sum of all edge capacities; here it is also assumed that the embedding and the parameter g are given in the input. Finally, we remark that multiterminal flow [21] and Wiener index computations [10] have exploited the treewidth parameter, assuming that the corresponding tree decomposition of the graph is given. However, in both publications [21, 10] the dependency on the parameter k is *exponential*.

Our contribution

In this paper, for illustrating the potential algorithmic challenges posed by the "FPT inside P" framework (which seem to go clearly beyond the known "FPT inside P" examples), we focus on Longest Path on Interval Graphs, which is known to be solvable in $O(n^4)$ time [23], and we derive an PL-FPT-algorithm (with the appropriate parameterization) that satisfies all three desirable algorithmic properties described above.

On general graphs, the decision variant of Longest Path is NP-complete and many FPT algorithms have been designed for it, e.g., [5, 13, 26, 35], contributing to the parameterized algorithm design toolkit techniques such as color-coding [5] (and further randomized techniques [13, 26]) as well as algebraic approaches [35]. Longest Path is known to be solvable in polynomial time only on very few non-trivial graph classes [23, 29] (see also [33] for much smaller graph classes). In particular, a few years ago it was shown that Longest Path on Interval Graphs can be solved in polynomial time, providing an algorithm that runs in $O(n^4)$ time [23]. Notably, a longest path in a *proper* interval graph can be computed by a *trivial* linear-time algorithm since every connected proper interval graph has a Hamiltonian path [7]. Consequently, as these two graph classes seem to behave quite differently, it is natural to parameterize Longest Path on Interval Graphs by the size k of a *minimum proper interval (vertex) deletion set*, i.e., by the minimum number of vertices that need to be deleted to obtain a proper interval graph. That is, this parameterization exploits what is also known as "distance from triviality" [20, 17] in the sense that the parameter k measures how far a given input instance is from a trivially solvable special case. As it turns out, one can compute a 4-approximation of k in $O(n + m)$ time for an interval graph with n vertices and m edges. Based on this, we provide a polynomial fixed-parameter algorithm that runs in $O(k^9 n)$ time, thus proving that Longest Path on Interval Graphs is in the class PL-FPT when parameterized by the size of a minimum proper interval deletion set.

To develop our algorithm, we first introduce in Section 2 two data reduction rules on interval graphs. Each of these reductions shrinks the size of specific vertex subsets, called *reducible* and *weakly reducible* sets, respectively. Then, given any proper interval deletion set D of an interval graph G, in Section 3 we appropriately decompose the graph $G \setminus D$ into two collections $\mathcal{S}_1$ and $\mathcal{S}_2$ of reducible and weakly reducible sets, respectively, on which we apply the reduction rules of Section 2. The resulting interval graph $\widehat{G}$ is *weighted* (with weights on its vertices) and has some special properties; we call $\widehat{G}$ a *special weighted interval*

graph with *parameter* κ, where in this case $\kappa = O(k^3)$. Notably, although $\widehat{G}$ has reduced size, it still has $O(n)$ vertices. Then, in Section 4 we present a fixed-parameter algorithm (with parameter κ) computing in $O(\kappa^3 n)$ time the maximum weight of a path in a special weighted interval graph. We note here that such a maximum-weight path in a special weighted interval graph can be directly mapped back to a longest path in the original interval graph. Thus, our algorithm computes a longest path in the initial interval graph G in $O(\kappa^3 n) = O(k^9 n)$ time. In the concluding section, we give a brief outlook.

Notation

We consider finite, simple, and undirected graphs. Given a graph G, we denote by $V(G)$ and $E(G)$ the sets of its vertices and edges, respectively. A graph G is *weighted* if it is given along with a weight function $w : V(G) \to \mathbb{N}$ on its vertices. An edge between two vertices u and v of a graph $G = (V, E)$ is denoted by uv, and in this case u and v are said to be *adjacent*. The *neighborhood* of a vertex $u \in V$ is the set $N(u) = \{v \in V : uv \in E\}$ of its adjacent vertices. Furthermore we denote by $G[S]$ the subgraph of G induced by the vertex set S and we define $G \setminus S = G[V \setminus S]$. A set $S \subseteq V$ induces an *independent set* (resp. a *clique*) in G if $uv \notin E$ (resp. if $uv \in E$) for every pair of vertices $u, v \in S$.

A graph $G = (V, E)$ is an *interval* graph if each vertex $v \in V$ can be bijectively assigned to a closed interval I_v on the real line, such that $uv \in E$ if and only if $I_u \cap I_v \neq \emptyset$, and then the collection of intervals $\mathcal{I} = \{I_v : v \in V\}$ is an *interval representation* of G. The graph G is a *proper interval graph* if it admits an interval representation $\mathcal{I}$ such that $I_u \not\subseteq I_v$ for every $u, v \in V$, and then $\mathcal{I}$ is a *proper interval representation*. Given an interval graph G, a subset $D \subseteq V(G)$ is a *proper interval deletion set* of G if $G \setminus D$ is a proper interval graph. The *proper interval deletion number* of G is the size of the smallest proper interval deletion set. Finally, for any positive integer t, we denote $[t] = \{1, 2, \ldots, t\}$.

2 Data reductions on interval graphs

In this section we present two data reductions on interval graphs. The first reduction (cf. Reduction Rule 1) shrinks the size of a collection of vertex subsets of a certain kind, called *reducible* sets, and it produces a *weighted* interval graph. The second reduction (cf. Reduction Rule 2) is applied to an arbitrary weighted interval graph; it shrinks the size of a collection of another kind of vertex subsets, called *weakly reducible* sets, and it produces a smaller weighted interval graph. Both reductions retain as an invariant the maximum path weight.

In the remainder of the paper we assume that we are given an interval graph G with n vertices and m edges as input, together with an interval representation $\mathcal{I}$ of G, where the endpoints of the intervals are given sorted increasingly. Without loss of generality we assume that the endpoints of all intervals are distinct. For every vertex $v \in V(G)$ we denote by $I_v = [l_v, r_v]$ the interval of $\mathcal{I}$ that corresponds to v, i.e., l_v and r_v are the left and the right endpoint of I_v, respectively. In particular, G is assumed to be given along with the *right-endpoint ordering* σ of its vertices $V(G)$, i.e., $u <_\sigma v$ if and only if $r_u < r_v$ in the interval representation $\mathcal{I}$. Given a set $S \subseteq V(G)$ we denote by $\mathcal{I}[S]$ the interval representation induced from $\mathcal{I}$ on the intervals of the vertices of S. Finally we denote by $\mathbf{span}(S)$ the interval $[\min\{l_v : v \in S\}, \max\{r_v : v \in S\}]$.

It is well-known that an interval graph G is a proper interval graph if and only if G is $K_{1,3}$-free, i.e., if G does not include the claw $K_{1,3}$ with four vertices (cf. Figure 1) as an induced subgraph [32]. It is worth noting here that it is unknown whether a minimum proper interval deletion set of an interval graph G can be computed in polynomial time. However,

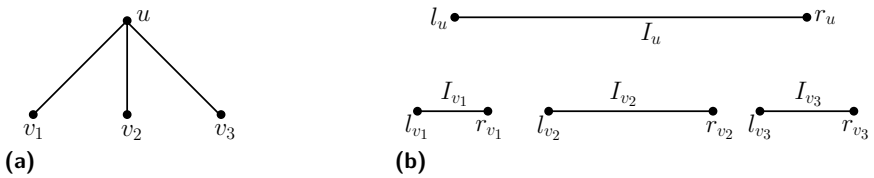


Figure 1 (a) The forbidden induced subgraph (claw $K_{1,3}$) for an interval graph to be a proper interval graph and (b) an interval representation of the $K_{1,3}$.

since there is a unique forbidden induced subgraph $K_{1,3}$ on four vertices, we can apply Cai's generic algorithm [11] on an arbitrary given interval graph G with n vertices to compute a proper interval deletion set of G with minimum size k in FPT time $O(4^k \cdot \text{poly}(n))$. As we prove in the next theorem, a 4-approximation of the minimum proper interval deletion number of an interval graph can be computed much more efficiently.

▶ **Theorem 1.** *Let $G = (V, E)$ be an interval graph, where $|V| = n$ and $|E| = m$. Let k be the size of the minimum proper interval deletion set of G. Then a proper interval deletion set of size at most $4k$ can be computed in $O(n + m)$ time.*

Note that, whenever four vertices induce a claw $K_{1,3}$ in an interval graph G, then in the interval representation $\mathcal{I}$ of G at least one of these intervals is necessarily properly included in another one (e.g., $I_{v_2} \subseteq I_u$ in Figure 1b). However the converse is not always true, as there may exist two vertices u, v in G such that $I_v \subseteq I_u$, although u and v do not participate in any claw $K_{1,3}$ in G. An interval representation $\mathcal{I}$ is called *semi-proper* if, whenever $I_v \subseteq I_u$ in $\mathcal{I}$, then the vertices u and v participate in a claw $K_{1,3}$ in G. Every interval representation $\mathcal{I}$ of a graph G can be efficiently transformed into a semi-proper representation $\mathcal{I}'$ of G, as we prove in the next theorem. In the remainder of the paper we always assume that this preprocessing step has been already applied to $\mathcal{I}$.

▶ **Theorem 2** (preprocessing). *Given an interval representation $\mathcal{I}$, a semi-proper interval representation $\mathcal{I}'$ can be computed in $O(n + m)$ time.*

All our results on interval graphs rely on the notion of a *normal* path [23] (also referred to as a *straight* path in [15, 25]). This notion has also been extended to the greater class of cocomparability graphs [29]. Normal paths are useful in the analysis of our data reductions in this section, as well as in our algorithm in Section 4, as they impose certain *monotonicity* properties of the paths. Informally, the vertices in a normal path appear in a "left-to-right fashion" in the right-endpoint ordering σ. It is known that, for every path P of an interval graph G, there exists a normal path P' on the same vertices as P [23].

▶ **Definition 3.** Let $G = (V, E)$ be an interval graph and σ be a right-endpoint ordering of V. The path $P = (v_1, v_2, \ldots, v_k)$ of G is *normal* if v_1 is the leftmost vertex of $V(P)$ in σ, and if v_i is the leftmost vertex of $N(v_{i-1}) \cap \{v_i, v_{i+1}, \ldots, v_k\}$ in σ, for every $i = 2, \ldots, k$.

The first data reduction

Before we present our Reduction Rule 1, first we introduce the notion of a *reducible* set of vertices.

▶ **Definition 4.** Let G be a (weighted) interval graph and $\mathcal{I}$ be an interval representation of G. A set $S \subseteq V(G)$ is *reducible* if it satisfies the following:

1. $\mathcal{I}[S]$ induces a connected proper interval representation of $G[S]$ and
2. for every $v \in V(G)$ such that $I_v \subseteq \mathbf{span}(S)$ it holds $v \in S$.

The intuition behind reducible sets is as follows. For every reducible set S, a longest path P contains either all vertices of S or none of them. Furthermore, in a longest path P which is *normal* and contains the whole set S, the vertices of S appear *consecutively* in P. Thus we can reduce the number of vertices in a longest normal path P (without changing its total weight) by replacing all vertices of S with a single vertex having weight $|S|$. Now we reduce the interval graph G to the *weighted* interval graph $G^{\#}$ with fewer vertices.

▶ **Reduction Rule 1** (first data reduction). *Let $G = (V, E)$ be an interval graph, $\mathcal{I}$ be an interval representation of G, and D be a proper interval deletion set of G. Let $\mathcal{S}$ be a set of vertex disjoint reducible sets of G, where $S \cap D = \emptyset$, for every $S \in \mathcal{S}$. The* weighted interval graph $G^{\#} = (V^{\#}, E^{\#})$ *is induced by the* weighted interval representation $\mathcal{I}^{\#}$*, which is derived from $\mathcal{I}$ as follows:*

- *for every $S \in \mathcal{S}$, replace in $\mathcal{I}$ the intervals $\{I_v : v \in S\}$ with the single interval $I_S = \mathbf{span}(S)$ which has weight $|S|$; all other intervals receive weight* 1.

In the next theorem we state the correctness of Reduction Rule 1.

▶ **Theorem 5.** *Let ℓ be a positive integer. Then the longest path in G has ℓ vertices if and only if the maximum weight of a path in $G^{\#}$ is ℓ.*

The second data reduction

Before we present our Reduction Rule 2, we introduce the notion of a *weakly reducible* set.

▶ **Definition 6.** Let G be a (weighted) interval graph and $\mathcal{I}$ be an interval representation of G. A set $S \subseteq V(G)$ is *weakly reducible* if it satisfies the following:

1. $\mathcal{I}[S]$ induces a connected proper interval representation of $G[S]$ and
2. for every $v \in V(G)$ and every $u \in S$, if $I_v \subseteq I_u$, then $S \subseteq N(v)$.

The intuition behind weakly reducible sets is as follows. For every weakly reducible set S, a longest path P contains either all vertices of S or none of them. Furthermore, in a longest path P which is *normal* and contains the whole set S, the appearance of the vertices of S in P is *interrupted at most* $|D| + 3$ times by vertices outside S. That is, such a path P has at most $\min\{|S|, |D| + 4\}$ vertex-maximal subpaths with vertices from S. Thus we can reduce the number of vertices in a maximum-weight normal path P (without changing its total weight) by replacing all vertices of S with $\min\{|S|, |D| + 4\}$ vertices; each of these new vertices has the same weight and their total weight sums up to $|S|$. Now we reduce the weighted interval graph G to the weighted interval graph $\widehat{G}$ with fewer vertices.

▶ **Reduction Rule 2** (second data reduction). *Let G be a weighted interval graph with weight function $w : V(G) \to \mathbb{N}$ and $\mathcal{I}$ be an interval representation of G. Let D be a proper interval deletion set of G. Finally, let $\mathcal{S} = \{S_1, S_2, \ldots, S_{|\mathcal{S}|}\}$ be a family of pairwise disjoint weakly reducible sets, where $S_i \cap D = \emptyset$ for every $i \in [|\mathcal{S}|]$. We recursively define the graphs $G_0, G_1, \ldots, G_{|\mathcal{S}|}$ with the interval representations $\mathcal{I}_0, \mathcal{I}_1, \ldots, \mathcal{I}_{|\mathcal{S}|}$ as follows:*

- *$G_0 = G$ and $\mathcal{I}_0 = \mathcal{I}$,*
- *for $1 \leq i \leq |\mathcal{S}|$, $\mathcal{I}_i$ is obtained by replacing in $\mathcal{I}_{i-1}$ the intervals $\{I_v : v \in S_i\}$ with $\min\{|S_i|, |D|+4\}$ copies of the interval $I_{S_i} = \mathbf{span}(S_i)$, each of them having equal weight $\frac{1}{\min\{|S_i|,|D|+4\}} \sum_{u \in S_i} w(u)$; all other intervals remain unchanged, and*
- *finally $\widehat{G} = G_{|\mathcal{S}|}$ and $\widehat{\mathcal{I}} = \mathcal{I}_{|\mathcal{S}|}$.*

Note that in the construction of the interval representation $\mathcal{I}_i$ by Reduction Rule 2, where $i \in [|\mathcal{S}|]$, we can always slightly perturb the endpoints of the $\min\{|S_i|, |D|+4\}$ copies of the interval $I_{S_i} = \mathbf{span}(S_i)$ such that all endpoints remain distinct in $\mathcal{I}_i$, and such that these $\min\{|S_i|, |D|+4\}$ newly introduced intervals induce a proper interval representation in $\mathcal{I}_i$. We are now ready to prove the correctness of Reduction Rule 2.

▶ **Theorem 7.** *Let ℓ be a positive integer. Then the maximum weight of a path in G is ℓ if and only if the maximum weight of a path in $\widehat{G}$ is ℓ.*

3 Special weighted interval graphs

In this section we sequentially apply the two data reductions of Section 2 to a given interval graph G with a proper interval deletion set D. To do so, first we appropriately define a specific family $\mathcal{S}_1$ of *reducible* sets in $G \setminus D$ and we apply Reduction Rule 1 to G with respect to the family $\mathcal{S}_1$, resulting in the weighted interval graph $G^{\#}$. After this operation, D remains a proper interval deletion set of the graph $G^{\#}$. Then we appropriately define a family $\mathcal{S}_2$ of *weakly reducible* sets in $G^{\#} \setminus D$ and we apply Reduction Rule 2 to $G^{\#}$ with respect to the family $\mathcal{S}_2$, resulting to the weighted interval graph $\widehat{G}$. As it turns out, the vertex sets of the union $\mathcal{S}_1 \cup \mathcal{S}_2$ of these two families are a partition of the initial graph $G \setminus D$. Furthermore, Theorems 5 and 7 imply that the longest path in the initial graph G has ℓ vertices if and only if the maximum weight of a path in the final weighted graph $\widehat{G}$ is ℓ. The graph $\widehat{G}$ is then given as input to our parameterized algorithm of Section 4. Now we introduce the crucial notion of a *special weighted interval graph* with *parameter* κ.

▶ **Definition 8** (special weighted interval graph with parameter κ)**.** Let $G = (V, E)$ be a weighted interval graph, $\mathcal{I}$ be an interval representation of G, and $\kappa \in \mathbb{N}$, where V can be partitioned into two sets A and B such that:

1. A is an independent set in G,
2. for every $v \in A$ and every $u \in V \setminus \{v\}$, we have $I_u \not\subseteq I_v$ in $\mathcal{I}$, and
3. $|B| \leq \kappa$.

Then G (resp. $\mathcal{I}$) is a *special weighted interval graph* (resp. *representation*) with *parameter* κ. The partition $V = A \cup B$ is a *special vertex partition* of G.

As we prove in the next theorem, the graph $\widehat{G}$ is a special weighted interval graph and it can be computed efficiently.

▶ **Theorem 9.** *Let $G = (V, E)$ be an interval graph, where $|V| = n$. Let D be a proper interval deletion set of G, where $|D| = k$. Then the graph $\widehat{G} = (\widehat{V}, \widehat{E})$ is a* special weighted interval graph *with* parameter $\kappa = O(k^3)$*. Furthermore, $\widehat{G}$ and a special vertex partition $\widehat{V} = A \cup B$ of $\widehat{G}$ can be computed in $O(k^2 n)$ time.*

Note that, although $\widehat{G}$ is a special weighted interval graph with a parameter κ that depends only on the size of D, it may still have $O(n)$ vertices, as the independent set A in the special vertex partition may be arbitrarily large.

4 Parameterized Longest Path on Interval Graphs

In this section, we first present Algorithm 1 which computes in $O(\kappa^3 n)$ time the maximum weight of a path in a *special weighted* interval graph with parameter κ. Then, using Algorithm 1 and the results of Sections 2 and 3, we conclude with our fixed-parameter algorithm for Longest Path on Interval Graphs, where the parameter k is the size of a minimum proper interval deletion set D.

The algorithm for special weighted interval graphs

Consider a *special weighted interval graph* $G = (V, E)$ with parameter $\kappa \in \mathbb{N}$ and $|V| = n$, which is given along with a special interval representation $\mathcal{I}$ and a special vertex partition $V = A \cup B$. Recall from Definition 8 that A is an independent set and that $|B| \leq \kappa$. Let $w : V \to \mathbb{N}$ be the vertex weight function of G. For simplicity, we add to the set B an isolated dummy vertex v_0 such that $v_0 <_\sigma v_1$ and $w(v_0) = 0$ (cf. the first line of Algorithm 1). For every vertex $v \in B$, we define:

$$\xi_v = \begin{cases} l_u & \text{if } l_v \in I_u \text{ for some } u \in A, \\ l_v & \text{otherwise.} \end{cases} \tag{1}$$

Since A is an independent set by assumption, for every $v \in B$ there exists at most one $u \in A$ such that $l_v \in I_u$. Thus ξ_v is well-defined. Now we define the set $\Xi = \{\xi_v, l_v : v \in B\}$. Note that $|\Xi| = O(\kappa)$. Furthermore, for every $u, v \in V$, where $u \in N(v)$ and $u <_\sigma v$, we define the vertex $\pi_{u,v} = \max_\sigma\{\{u\} \cup \{w \in B \cap N(u) : u <_\sigma w <_\sigma v\}\}$. Note that, by definition, if $\pi_{u,v} \neq u$ then $\pi_{u,v} \in B$. Furthermore, due to the condition that $u \in N(v)$ in the definition of $\pi_{u,v}$, it follows that $u \in B$ or $v \in B$, since A is an independent set. That is, vertex $\pi_{u,v}$ is defined for at most $O(\kappa n)$ pairs of vertices u, v. Now we provide the crucial definition of the subgraphs $G_\xi(v_i)$ of G, for every $\xi \in \Xi$ and $i \in [n]$ such that $\xi < r_{v_i}$.

▶ **Definition 10.** Let $\xi \in \Xi$ and $i \in [n]$ such that $\xi < r_{v_i}$. We define the induced subgraph $G_\xi(v_i) = G[\{v \in V : \xi \leq l_v < r_v \leq r_{v_i}\}]$ of G which contains all vertices whose intervals are entirely contained between the points ξ and r_{v_i}.

▶ **Notation 1.** *Let $\xi \in \Xi$ and $i \in [n]$ such that $\xi < r_{v_i}$. Furthermore let $y \in V(G_\xi(v_i))$ such that $y \in N(v_i)$. We denote by $P_\xi(v_i, y)$ a* maximum-weight normal path *of $G_\xi(v_i)$, among those normal paths whose last vertex is y. For every path $P_\xi(v_i, y)$, we denote its weight $w(P_\xi(v_i, y))$ by $W_\xi(v_i, y)$.*

Note that, by Definition 10, if $l_{v_i} < \xi$ then the vertex v_i does not belong to the subgraph $G_\xi(v_i)$. Now we state three lemmas that are crucial for the correctness of Algorithm 1.

▶ **Lemma 11.** *Let $\xi \in \Xi$ and $i \in [n]$, where $\xi < r_{v_i}$, and let $y \in V(G_\xi(v_i))$ and $y \in N(v_i)$. If $l_y < l_{v_i}$ or $v_i \notin V(G_\xi(v_i))$, then $W_\xi(v_i, y) = W_\xi(\pi_{y,v_i}, y)$.*

▶ **Lemma 12.** *Let $\xi \in \Xi$ and $i \in [n]$, where $\xi < r_{v_i}$, and let $v_i \in V(G_\xi(v_i))$. Then $P_\xi(v_i, v_i) = (P_1, v_i)$, where $w(P_1) = \max\{W_\xi(\pi_{x,v_i}, x) : x \in V(G_\xi(v_i)), l_{v_i} < r_x < r_{v_i}\}$.*

▶ **Lemma 13.** *Let $\xi \in \Xi$ and $i \in [n]$, where $\xi < r_{v_i}$, and let $v_i, y \in V(G_\xi(v_i))$ and $y \in N(v_i)$. Let $P_\xi(v_i, y) = (P_1, v_i, P_2)$. If $P_2 \neq (y)$ then there exists some $\zeta \in \Xi$, where $l_{v_i} < \zeta \leq l_y$, such that*

$$w(P_1) = \max\{W_\xi(\pi_{x,v_i}, x) : x \in V(G_\xi(v_i)), l_{v_i} < r_x < \zeta\}, \tag{2}$$

$$w(P_2) = W_\zeta(\pi_{y,v_i}, y). \tag{3}$$

Otherwise, if $P_2 = (y)$ then $l_{v_i} < l_y$ and

$$w(P_1) = \max\{W_\xi(\pi_{x,v_i}, x) : x \in V(G_\xi(v_i)), l_{v_i} < r_x < l_y\}. \tag{4}$$

We are now ready to present Algorithm 1, which computes the maximum weight of a path in a given *special weighted* interval graph G. It is easy to check that Algorithm 1 can be slightly modified such that it returns the actual path P instead of its weight only.

Algorithm 1 Computing a maximum-weight path of a special interval graph

Input: A special weighted interval graph $G = (V, E)$ with parameter $\kappa \in \mathbb{N}$, along with the special interval representation $\mathcal{I}$ of G and the partition $V = A \cup B$, where $\sigma = (v_1, v_2, \ldots, v_n)$ is a right-endpoint ordering of V.

Output: The maximum weight of a path in G

1: Add an isolated dummy vertex v_0 with $w(v_0) = 0$ to set B, where $v_0 <_\sigma v_1$

2: **for** $i = 0$ to n **do**

3: **for** every $\xi \in \Xi$ where $\xi < r_{v_i}$ **do**

4: **if** $v_i \in V(G_\xi(v_i))$ **then** $W_\xi(v_i, v_i) \leftarrow w(v_i)$ {initialization}

5: **for** every $y \in V(G_\xi(v_i))$ where $y \in N(v_i)$ **do**

6: $W_\xi(v_i, y) \leftarrow W_\xi(\pi_{y,v_i}, y)$ {initialization}

7: **if** $v_i \in V(G_\xi(v_i))$ **then**

8: $W_1 \leftarrow \max\{W_\xi(\pi_{x,v_i}, x) : x \in V(G_\xi(v_i)), l_{v_i} < r_x < r_{v_i}\}$

9: $W_\xi(v_i, v_i) \leftarrow \max\{W_\xi(v_i, v_i), W_1 + w(v_i)\}$

10: **for** every $y \in V(G_\xi(v_i))$ where $y \in N(v_i)$ **do**

11: **if** $l_y < l_{v_i}$ or $v_i \notin V(G_\xi(v_i))$ **then**

12: $W_\xi(v_i, y) \leftarrow W_\xi(\pi_{y,v_i}, y)$

13: **else**

14: $W'_1 \leftarrow \max\{W_\xi(\pi_{x,v_i}, x) : x \in V(G_\xi(v_i)), l_{v_i} < r_x < l_y\}$

15: **for** every $\zeta \in \Xi$ with $l_{v_i} < \zeta \leq l_y$ **do**

16: $W_1 \leftarrow \max\{W_\xi(\pi_{x,v_i}, x) : x \in V(G_\xi(v_i)), l_{v_i} < r_x < \zeta\}$

17: $W_\xi(v_i, y) \leftarrow \max\{W_\xi(v_i, y), W_1 + w(v_i) + W_\zeta(\pi_{y,v_i}, y)\}$

18: $W_\xi(v_i, y) \leftarrow \max\{W_\xi(v_i, y), W'_1 + w(v_i) + w(y)\}$

19: **return** $\max\{W_{l_{v_0}}(v_i, v_i), W_{l_{v_0}}(v_i, y) : 1 \leq i \leq n,\ y <_\sigma v_i,\ y \in N(v_i)\}$

First we give a brief overview of the algorithm. Using dynamic programming, it computes a 3-dimensional table. In this table, for every point $\xi \in \Xi$, every index $i \in [n]$, and every vertex $y \in V(G_\xi(v_i))$, where $\xi < r_{v_i}$ and $y \in N(v_i)$, the entry $W_\xi(v_i, y)$ (resp. the entry $W_\xi(v_i, v_i)$) keeps the weight of a normal path in the subgraph $G_\xi(v_i)$ which is the largest among those normal paths whose last vertex is y (resp. v_i). Thus, since $w(v_0) = 0$ for the dummy isolated vertex v_0 (cf. line 1 of the algorithm), the maximum weight of a path in G will be eventually stored in one of the entries $\{W_{l_{v_0}}(v_i, v_i) : 1 \leq i \leq n\}$ or in one of the entries $\{W_{l_{v_0}}(v_i, y) : 1 \leq i \leq n,\ y <_\sigma v_i,\ y \in N(v_i)\}$, depending on whether the last vertex y of the desired maximum-weight path coincides with the rightmost vertex v_i of this path in the ordering σ (cf. line 19 of the algorithm).

Note that for every computed entry $W_\xi(v_i, y)$ the vertices v_i and y are adjacent, and thus $v_i \in B$ or $y \in B$, since A is an independent set. Thus, since $|B| = O(\kappa)$, there are at most $O(\kappa n)$ such eligible pairs of vertices v_i, y. Furthermore, since also $|\Xi| = O(\kappa)$, the computed 3-dimensional table stores at most $O(\kappa^2 n)$ entries $W_\xi(v_i, v_i)$ and $W_\xi(v_i, y)$. From the *for*-loops of lines 2-3 of the algorithm and from the obvious inductive hypothesis we may assume that during the $\{i, \xi\}$th iteration all previous values $W_{\xi'}(v_{i'}, v_{i'})$ and $W_{\xi'}(v_{i'}, y')$, where $i' < i$ or $\xi' < \xi$, have been correctly computed at a previous iteration.

In the initialization phase for a particular pair $\{i, \xi\}$ (cf. lines 4-6) the algorithm computes some initial values for $W_\xi(v_i, v_i)$ and $W_\xi(v_i, y)$. For a path with v_i as its last vertex, we are

only interested in the case where $v_i \in V(G_\xi(v_i))$; in this case we initialize $W_\xi(v_i, v_i) = w(v_i)$, cf. line 4. For a path with $y \neq v_i$ as its last vertex (cf. lines 5-6), we initialize $W_\xi(v_i, y) = W_\xi(\pi_{y,v_i}, y)$, since the path $P_\xi(\pi_{y,v_i}, y)$ is indeed a normal path of the graph $G_\xi(\pi_{y,v_i})$, which is an induced subgraph of $G_\xi(v_i)$.

For the induction step phase (cf. lines 7-18) the algorithm updates the initialized entries $W_\xi(v_i, v_i)$ and $W_\xi(v_i, y)$ according to Lemmas 11-13. To update the value $W_\xi(v_i, v_i)$ we only need to consider the case where $v_i \in V(G_\xi(v_i))$; in this case $W_\xi(v_i, v_i)$ is updated in lines 7-9 according to Lemma 12. The values of $W_\xi(v_i, y)$, where $y \neq v_i$, are updated in lines 10-18. In particular, in the case where $l_y < l_{v_i}$ or $v_i \notin V(G_\xi(v_i))$, the value of $W_\xi(v_i, y)$ is updated in lines 11-12 according to Lemma 11. Otherwise, $W_\xi(v_i, y)$ is updated in lines 14-18 according to Lemma 13.

▶ **Theorem 14.** *Let $G = (V, E)$ be a special weighted interval graph with n vertices and parameter κ. Then Algorithm 1 computes the maximum weight of a path P in G in $O(\kappa^3 n)$ time.*

The general algorithm

Here we present our *parameterized linear-time* algorithm for Longest Path on Interval Graphs, whose running time has a *polynomial dependency* on k.

▶ **Theorem 15.** *Let $G = (V, E)$ be an interval graph, where $|V| = n$ and $|E| = m$, and let k be the minimum size of a proper interval deletion set of G. Let $\mathcal{I}$ be an interval representation of G whose endpoints are sorted increasingly. Then:*

1. *a proper interval deletion set D, where $|D| \leq 4k$, can be computed in $O(n + m)$ time,*
2. *a semi-proper interval representation $\mathcal{I}'$ can be constructed in $O(n + m)$ time,*
3. *given D and $\mathcal{I}'$, a longest path of G can be computed in $O(k^9 n)$ time.*

5 Outlook

Our work heads at stimulating a general research program which systematically exploits the concept of fixed-parameter tractability for polynomially solvable problems. For several fundamental and widely known problems, the time complexity of the currently fastest algorithms are upper-bounded by polynomials of large degrees. One of the most prominent examples is arguably the celebrated polynomial-time recognition algorithm for *perfect graphs*, whose time complexity still remains $O(n^9)$ [14]. Apart from trying to improve the *worst-case* time complexity for such problems, which may be a very difficult (if not impossible) task, the complementary approach that we propose here is to try to detect the *parameter* that causes these high-degree polynomial-time algorithms and to separate the dependency of the time complexity from this parameter such that the dependency on the input size becomes as close to linear as possible. We believe that the "FPT inside P" field is very rich and offers plenty of research possibilities.

We conclude with two related topics that may lead to further interactions. First, we remark that in classical parameterized complexity analysis there is a growing awareness concerning the polynomial-time factors that often have been neglected [34]. Notably, there are some prominent fixed-parameter tractability results giving *linear-time* factors in the input size (but quite large exponential factors in the parameter); these include Bodlaender's famous algorithm for computing treewidth [8] and the more recent algorithm for computing the crossing number of a graph [24]. Interestingly, these papers emphasize "linear time" in their titles, instead of "fixed-parameter tractability". In this spirit, our result for Longest Path

in Interval graphs is a "linear-time" algorithm where the dependency on the parameter is not exponential [8, 24] but *polynomial*.

Second, polynomial-time solvability and the corresponding lower bounds have been of long-standing interest, e.g., it is believed that the famous 3SUM problem is only solvable in quadratic time and this conjecture has been employed for proving relative lower bounds for other problems [19]. Very recently, there was a significant push for this research direction with many new relative lower bounds [2, 1, 9]. The "FPT inside P" approach might help in "breaking" this nonlinear relative lower bounds by introducing useful parameterizations and striving for PL-FPT results, whenever possible; some very recent results in this direction concern the problems of computing the diameter and the radius in a graph [3].

References

1 Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, APSP and diameter. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1681–1697, 2015.

2 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Proceedings of the 55th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 434–443, 2014.

3 Amir Abboud, Virginia Vassilevska Williams, and Joshua Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2016. To appear.

4 Ittai Abraham, Amos Fiat, Andrew V. Goldberg, and Renato Fonseca F. Werneck. Highway dimension, shortest paths, and provably efficient algorithms. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 782–793, 2010.

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

6 Amihood Amir, Moshe Lewenstein, and Ely Porat. Faster algorithms for string matching with k mismatches. *Journal of Algorithms*, 50(2):257–275, 2004.

7 Alan A. Bertossi. Finding Hamiltonian circuits in proper interval graphs. *Information Processing Letters*, 17(2):97–101, 1983.

8 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. on Comp.*, 25(6):1305–1317, 1996.

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

10 Sergio Cabello and Christian Knauer. Algorithms for graphs of bounded treewidth via orthogonal range searching. *Computational Geometry*, 42(9):815–824, 2009.

11 Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996.

12 Erin W. Chambers, Jeff Erickson, and Amir Nayyeri. Homology flows, cohomology cuts. *SIAM J. on Comput.*, 41(6):1605–1634, 2012.

13 Jianer Chen, Songjian Lu, Sing-Hoi Sze, and Fenghui Zhang. Improved algorithms for path, matching, and packing problems. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 298–307, 2007.

14 Maria Chudnovsky, Gérard Cornuéjols, Xinming Liu, Paul Seymour, and Kristina Vušković. Recognizing Berge graphs. *Combinatorica*, 25(2):143–186, 2005.

15 Peter Damaschke. Paths in interval graphs and circular arc graphs. *Discrete Mathematics*, 112(1-3):49–64, 1993.

16 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

17 Michael R. Fellows, Bart M. P. Jansen, and Frances A. Rosamond. Towards fully multivariate algorithmics: Parameter ecology and the deconstruction of computational complexity. *European Journal of Combinatorics*, 34(3):541–566, 2013.

18 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006.

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

20 Jiong Guo, Falk Hüffner, and Rolf Niedermeier. A structural view on parameterizing problems: Distance from triviality. In *Proceedings of the 1st International Workshop on Parameterized and Exact Computation (IWPEC)*, pages 162–173, 2004.

21 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.

22 Jan M. Hochstein and Karsten Weihe. Maximum s-t-flow with k crossings in $O(k^3 n \log n)$ time. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 843–847, 2007.

23 K. Ioannidou, George B. Mertzios, and Stavros D. Nikolopoulos. The longest path problem has a polynomial solution on interval graphs. *Algorithmica*, 61(2):320–341, 2011.

24 Ken-ichi Kawarabayashi and Bruce A. Reed. Computing crossing number in linear time. In *Proceedings of the 39th ACM Symp. on Th. of Comp. (STOC)*, pages 382–390, 2007.

25 J. M. Keil. Finding Hamiltonian circuits in interval graphs. *Information Processing Letters*, 20:201–206, 1985.

26 Joachim Kneis, Daniel Mölle, Stefan Richter, and Peter Rossmanith. Divide-and-color. In *Proceedings of the 32nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 58–67, 2006.

27 Ioannis Koutis, Gary L. Miller, and Richard Peng. A nearly-$m \log n$ time solver for SDD linear systems. In *Proceedings of the IEEE 52nd Symposium on Foundations of Computer Science (FOCS)*, pages 590–598, 2011.

28 Nimrod Megiddo. Linear programming in linear time when the dimension is fixed. *Journal of the ACM*, 31(1):114–127, 1984.

29 George B. Mertzios and Derek G. Corneil. A simple polynomial algorithm for the longest path problem on cocomparability graphs. *SIAM J. on Discr. Math.*, 26(3):940–963, 2012.

30 Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, 2006.

31 James B. Orlin, Kamesh Madduri, K. Subramani, and Matthew D. Williamson. A faster algorithm for the single source shortest path problem with few distinct positive lengths. *Journal of Discrete Algorithms*, 8(2):189–198, 2010.

32 F. S. Roberts. Indifference graphs. In F. Harary, editor, *Proof Techniques in Graph Theory*, pages 139–146. Academic Press, New York, 1969.

33 R. Uehara and Y. Uno. On computing longest paths in small graph classes. *International Journal of Foundations of Computer Science*, 18(5):911–930, 2007.

34 René van Bevern. *Fixed-Parameter Linear-Time Algorithms for NP-hard Graph and Hypergraph Problems Arising in Industrial Applications*. PhD thesis, Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany, 2014.

35 Ryan Williams. Finding paths of length k in $O^*(2^k)$ time. *Information Processing Letters*, 109(6):315–318, 2009.

Meta-kernelization using Well-structured Modulators*

Eduard Eiben, Robert Ganian, and Stefan Szeider

Algorithms and Complexity Group, TU Wien
Vienna, Austria

Abstract

Kernelization investigates exact preprocessing algorithms with performance guarantees. The most prevalent type of parameters used in kernelization is the solution size for optimization problems; however, also structural parameters have been successfully used to obtain polynomial kernels for a wide range of problems. Many of these parameters can be defined as the size of a smallest *modulator* of the given graph into a fixed graph class (i.e., a set of vertices whose deletion puts the graph into the graph class). Such parameters admit the construction of polynomial kernels even when the solution size is large or not applicable. This work follows up on the research on meta-kernelization frameworks in terms of structural parameters.

We develop a class of parameters which are based on a more general view on modulators: instead of size, the parameters employ a combination of rank-width and split decompositions to measure structure inside the modulator. This allows us to lift kernelization results from modulator-size to more general parameters, hence providing smaller kernels. We show (i) how such large but well-structured modulators can be efficiently approximated, (ii) how they can be used to obtain polynomial kernels for any graph problem expressible in Monadic Second Order logic, and (iii) how they allow the extension of previous results in the area of structural meta-kernelization.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, G.2.1 Combinatorics

Keywords and phrases Kernelization, Parameterized complexity, Structural parameters, Rank-width, Split decompositions

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.114

1 Introduction

Kernelization investigates exact preprocessing algorithms with performance guarantees. Similarly as in parameterized complexity analysis, in kernelization we study *parameterized problems*: decision problems where each instance I comes with a parameter k. A parameterized problem is said to admit a kernel of size $f : \mathbb{N} \to \mathbb{N}$ if every instance (I, k) can be reduced in polynomial time to an equivalent instance (called the *kernel*) whose size and parameter are bounded by $f(k)$. For practical as well as theoretical reasons, we are mainly interested in the existence of *polynomial kernels*, i.e., kernels whose size is polynomial in k. The study of kernelization has recently been one of the main areas of research in parameterized complexity, yielding many important new contributions to the theory.

The by far most prevalent type of parameter used in kernelization is the *solution size*. Indeed, the existence of polynomial kernels and the exact bounds on their sizes have been studied for a plethora of distinct problems under this parameter, and the rate of advancement

* The authors acknowledge support by the Austrian Science Fund (FWF, projects P26696 and W1255-N23).

10th International Symposium on Parameterized and Exact Computation (IPEC 2015).
Editors: Thore Husfeldt and Iyad Kanj; pp. 114–126

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

achieved in this direction over the past 10 years has been staggering. Important findings were also obtained in the area of *meta-kernelization* [4, 12, 18], which is the study of general kernelization techniques and frameworks used to establish polynomial kernels for a wide range of distinct problems.

In parameterized complexity analysis, an alternative to parameterization by solution size has traditionally been the use of *structural parameters*. But while parameters such as *treewidth* and the more general *rank-width* allow the design of FPT algorithms for a range of important problems, it is known that they cannot be used to obtain polynomial kernels for problems of interest. Instead, the structural parameters used for kernelization often take the form of the size of minimum *modulators* (a modulator of a graph is a set of vertices whose deletion puts the graph into a fixed graph class). Examples of such parameters include the size of a minimum vertex cover [11, 5] (modulators into the class of edgeless graphs) or of a minimum feedback vertex set [6, 17] (modulators into the class of forests). While such structural parameters are not as universal as the structural parameters used in the context of fixed-parameter tractability, these results nonetheless allow efficient preprocessing of instances where the solution size is large and for problems where solution size simply cannot be used (such as 3-coloring).

This paper follows up on the recent line of research which studies meta-kernelization in terms of structural parameters. Gajarský et al. [13] developed a meta-kernelization framework parameterized by the size of a modulator to the class of graphs of bounded treedepth on sparse graphs. Ganian et al. [15] independently developed a meta-kernelization framework using a different parameter based on rank-width and modular decompositions (see Subsection 2.4 for details). Our results build upon both of the aforementioned papers by fully subsuming the meta-kernelization framework of [15] and lifting the meta-kernelization framework of [13] to more general graph classes. The class of problems investigated in this paper are problems which can be expressed using *Monadic Second Order* (MSO) logic (see Subsection 2.5).

The parameters for our kernelization results are also based on modulators. However, instead of parameterizing by the *size* of the modulator, we instead measure the *structure* of the modulator through a combination of rank-width and split decompositions. Due to its technical nature, we postpone the definition of our parameter, the *well-structure number*, to Section 3; for now, let us roughly describe it as the number of sets one can partition a modulator into so that each set induces a graph with bounded rank-width and a simple neighborhood. We call modulators which satisfy our conditions *well-structured*. A less restricted variant of the well-structure number has recently been used to obtain meta-theorems for FPT algorithms on graphs of unbounded rank-width [10].

After formally introducing the parameter, in Section 4 we showcase its applications on the special case of generalizing the *vertex cover number* by considering well-structured modulators to edgeless graphs. While it is known that there exist MSO-definable problems which do not admit a polynomial kernel parameterized by the vertex cover number on general graphs, on graphs of bounded expansion this is no longer the case (as follows for instance from [13]). On the class of graphs of bounded expansion, we prove that every MSO-definable problem admits a linear kernel parameterized by the well-structure number for edgeless graphs. As a corollary of our approach, we also show that every MSO-definable problem admits a linear kernel parameterized by the well-structure number for the empty graph (without any restriction on the expansion). We remark that the latter result represents a direct generalization of the results in [15]. The proof is based on a combination of a refined version of the replacement techniques developed in [10] together with the annotation framework used in [15].

Before we can proceed to wider applications of our parameter in kernelization, it is first necessary to deal with the subproblem of finding a suitable well-structured modulator in

polynomial time. We resolve this question for well-structured modulators to a vast range of graph classes. In particular, in Subsection 5.1 we obtain a 3-approximation algorithm for finding well-structured modulators to acyclic graphs, and in the subsequent Subsection 5.2 we show how to approximate well-structured modulators to any graph class characterized by a finite set of forbidden induced subgraphs within a constant factor.

Section 6 then contains our most general result, Theorem 11, which is the key for lifting kernelization results from modulators to well-structured modulators. The theorem states that whenever a modulator to a graph class $\mathcal{H}$ can be used to poly-kernelize some MSO-definable problem, this problem also admits a polynomial kernel when parameterized by the well-structure number for $\mathcal{H}$ as long as well-structured modulators to $\mathcal{H}$ can be approximated in polynomial time. The remainder of Section 6 then deals with the applications of this theorem. Since the class of graphs of treedepth bounded by some fixed integer can be characterized by a finite set of forbidden induced subgraphs, we can use well-structured modulators to lift the results of [13] from modulators to well-structured modulators for all MSO-definable decision problems. Furthermore, by applying the *protrusion* machinery of [4, 18] we show that, in the case of bounded degree graphs, parameterization by a modulator to acyclic graphs (i.e., a feedback vertex set) allows the computation of a linear kernel for any MSO-definable decision problem. By our framework it then follows that such modulators can also be lifted to well-structured modulators.

2 Preliminaries

The set of natural numbers (that is, positive integers) will be denoted by $\mathbb{N}$. For $i \in \mathbb{N}$ we write $[i]$ to denote the set $\{1, \ldots, i\}$. If $\sim$ is an equivalence relation over a set A, then for $a \in A$ we use $[a]_\sim$ to denote the equivalence class containing a.

2.1 Graphs

We will use standard graph theoretic terminology and notation (cf. [8]). All graphs in this document are simple and undirected.

Given a graph $G = (V(G), E(G))$ and $A \subseteq V(G)$, we denote by $N(A)$ the set of neighbors of A in $V(G) \setminus A$; if A contains a single vertex v, we use $N(v)$ instead of $N(\{v\})$. We use V and E as shorthand for $V(G)$ and $E(G)$, respectively, when the graph is clear from context. $G - A$ denotes the subgraph of G obtained by deleting A. For $A \subseteq V(G)$ we use $G[A]$ to denote the subgraph of G induced by the set A.

2.2 Splits and Split-Modules

A *split* of a connected graph $G = (V, E)$ is a vertex bipartition $\{A, B\}$ of V such that every vertex of $A' = N(B)$ has the same neighborhood in $B' = N(A)$. The sets A' and B' are called *frontiers* of the split.

Let $G = (V, E)$ be a graph. To simplify our exposition, we will use the notion of *split-modules* instead of splits where suitable. A set $A \subseteq V$ is called a *split-module* of G if there exists a connected component $G' = (V', E')$ of G such that $\{A, V' \setminus A\}$ forms a split of G'. Notice that if A is a split-module then A can be partitioned into A_1 and A_2 such that $N(A_2) \subseteq A$ and for each $v_1, v_2 \in A_1$ it holds that $N(v_1) \cap (V' \setminus A) = N(v_2) \cap (V' \setminus A)$; A_1 is then called the frontier of A. For technical reasons, V and $\emptyset$ are also considered split-modules. We say that two disjoint split-modules $X, Y \subseteq V$ are *adjacent* if there exist $x \in X$ and $y \in Y$ such that x and y are adjacent. We use $\lambda(A)$ to denote the frontier of split-module A.

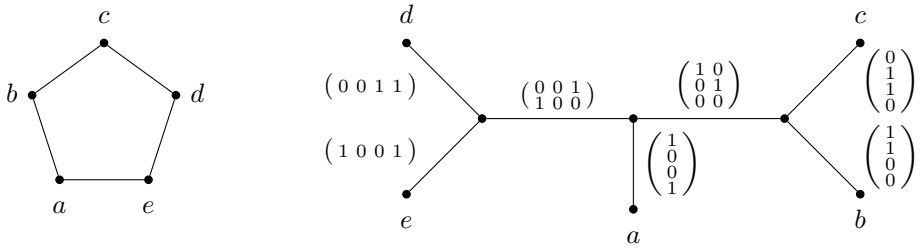


Figure 1 A rank-decomposition of the cycle C_5.

2.3 Rank-Width

For a graph G and $U, W \subseteq V(G)$, let $\boldsymbol{A}_G[U, W]$ denote the $U \times W$-submatrix of the adjacency matrix over the two-element field GF(2), i.e., the entry $a_{u,w}$, $u \in U$ and $w \in W$, of $\boldsymbol{A}_G[U, W]$ is 1 if and only if $\{u, w\}$ is an edge of G. The *cut-rank* function ρ_G of a graph G is defined as follows: For a bipartition (U, W) of the vertex set $V(G)$, $\rho_G(U) = \rho_G(W)$ equals the rank of $\boldsymbol{A}_G[U, W]$ over GF(2).

A *rank-decomposition* of a graph G is a pair (T, μ) where T is a tree of maximum degree 3 and $\mu : V(G) \to \{t : t \text{ is a leaf of } T\}$ is a bijective function. For an edge e of T, the connected components of $T - e$ induce a bipartition (X, Y) of the set of leaves of T. The *width* of an edge e of a rank-decomposition (T, μ) is $\rho_G(\mu^{-1}(X))$. The *width* of (T, μ) is the maximum width over all edges of T. The *rank-width* of G, $rw(G)$ in short, is the minimum width over all rank-decompositions of G. We denote by $\mathcal{R}_i$ the class of all graphs of rank-width at most i, and say that a graph class $\mathcal{H}$ is of *unbounded rank-width* if $\mathcal{H} \not\subseteq \mathcal{R}_i$ for any $i \in \mathbb{N}$.

▶ **Fact 1** ([16]). *Let $k \in \mathbb{N}$ be a constant and $n \geq 2$. For an n-vertex graph G, we can output a rank-decomposition of width at most k or confirm that the rank-width of G is larger than k in time $\mathcal{O}(n^3)$.*

More properties of rank-width can be found, for instance, in [21].

2.4 Fixed-Parameter Tractability and Kernels

We refer to the standard textbooks for basic notions in parameterized complexity such as *parameterized problem*, *kernelization* and *bikernelization* [9]. The following fact links the existence of bikernels to the existence of kernels.

▶ **Fact 2** ([1]). *Let $\mathcal{P}, Q$ be a pair of decidable parameterized problems such that Q is in* NP *and $\mathcal{P}$ is* NP*-complete. If there is a bikernelization from $\mathcal{P}$ to Q producing a polynomial bikernel, then $\mathcal{P}$ has a polynomial kernel.*

Within this paper, we will also consider (and compare to) various structural parameters which have been used to obtain polynomial kernels. We provide a brief overview of these parameters below.

A *modulator* of a graph G to a graph class $\mathcal{H}$ is a vertex set $X \subseteq V(G)$ such that $G - X \in \mathcal{H}$. We denote the cardinality of a minimum modulator to $\mathcal{H}$ in G by $mod^{\mathcal{H}}(G)$. The *vertex cover number* of a graph G ($vcn(G)$) is a special case of $mod^{\mathcal{H}}(G)$, specifically for $\mathcal{H}$ being the set of edgeless graphs. The vertex cover number has been used to obtain polynomial kernels for problems such as Largest Induced Subgraph [11] or Long Cycle along with other path and cycle problems [5]. Similarly, a *feedback vertex set* is a modulator to the class of acyclic graphs, and the size of a minimum feedback vertex set has been used to kernelize, for instance, Treewidth [6] or Vertex Cover [17].

For the final considered parameter, we will need the notion of *module*, which can be defined as a split-module with the restriction that every vertex in the split-module lies in its frontier. Then the *rank-width$_c$ cover number* [15] of a graph G ($rwc_c(G)$) is the smallest number of *modules* the vertex set of G can be partitioned into such that each module induces a subgraph of rank-width at most c. A wide range of problems, and in particular all MSO-definable problems, have been shown admit linear kernels when parameterized by the rank-width$_c$ cover number [15].

2.5 Monadic Second Order Logic on Graphs

We assume that we have an infinite supply of individual variables, denoted by lowercase letters x, y, z, and an infinite supply of set variables, denoted by uppercase letters X, Y, Z. *Formulas* of *monadic second-order logic* (MSO) are constructed from atomic formulas $E(x, y)$, $X(x)$, and $x = y$ using the connectives $\neg$ (negation), $\wedge$ (conjunction) and existential quantification $\exists x$ over individual variables as well as existential quantification $\exists X$ over set variables. Individual variables range over vertices, and set variables range over sets of vertices. The atomic formula $E(x, y)$ expresses adjacency, $x = y$ expresses equality, and $X(x)$ expresses that vertex x in the set X. From this, we define the semantics of monadic second-order logic in the standard way (this logic is sometimes called MSO_1).

Free and bound variables of a formula are defined in the usual way. A *sentence* is a formula without free variables. We write $\varphi(X_1, \ldots, X_n)$ to indicate that the set of free variables of formula φ is $\{X_1, \ldots, X_n\}$. If $G = (V, E)$ is a graph and $S_1, \ldots, S_n \subseteq V$ we write $G \models \varphi(S_1, \ldots, S_n)$ to denote that φ holds in G if the variables X_i are interpreted by the sets S_i, for $i \in [n]$. The problem framework we are mainly interested in is formalized below.

MSO Model Checking (MSO-MC$_\varphi$)
Instance: A graph G.
Question: Does $G \models \varphi$ hold?

While MSO model checking problems already capture many important graph problems, there are some well-known problems on graphs that cannot be captured in this way, such as Vertex Cover, Dominating Set, and Clique. Many such problems can be formulated in the form of *MSO optimization problems*. Let $\varphi = \varphi(X)$ be an MSO formula with one free set variable X and $\diamondsuit \in \{\leq, \geq\}$.

MSO-Opt$^{\diamondsuit}_{\varphi}$
Instance: A graph G and an integer $r \in \mathbb{N}$.
Question: Is there a set $S \subseteq V(G)$ such that $G \models \varphi(S)$ and $|S| \diamondsuit r$?

It is known that MSO formulas can be checked efficiently as long as the graph has bounded rank-width.

▶ **Fact 3** ([14]). *Let φ and $\psi = \psi(X)$ be fixed MSO formulas and let c be a constant. Then* MSO-MC$_\varphi$ *and* MSO-Opt$^{\diamondsuit}_{\varphi}$ *can be solved in $\mathcal{O}(n^3)$ time on the class of graphs of rank-width at most c, where n is the order of the input graph. Moreover, if G has rank-width at most c and and $S \subseteq V(G)$, it is possible to check whether $G \models \psi(S)$ in $\mathcal{O}(n^3)$ time.*

We review MSO *types* roughly following the presentation in [19]. The *quantifier rank* of an MSO formula φ is defined as the nesting depth of quantifiers in φ. For non-negative

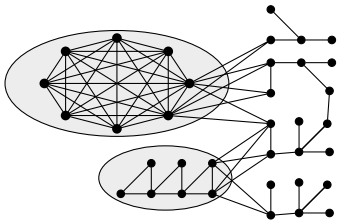


Figure 2 A graph with a $(2,1)$-well-structured modulator to forests (in the two shaded areas).

integers q and l, let $\mathrm{MSO}_{q,l}$ consist of all MSO formulas of quantifier rank at most q with free set variables in $\{X_1,\ldots,X_l\}$.

Let $\varphi = \varphi(X_1,\ldots,X_l)$ and $\psi = \psi(X_1,\ldots,X_l)$ be MSO formulas. We say φ and ψ are *equivalent*, written $\varphi \equiv \psi$, if for all graphs G and $U_1,\ldots,U_l \subseteq V(G)$, $G \models \varphi(U_1,\ldots,U_l)$ if and only if $G \models \psi(U_1,\ldots,U_l)$. Given a set F of formulas, let $F/\equiv$ denote the set of equivalence classes of F with respect to $\equiv$. A system of representatives of $F/\equiv$ is a set $R \subseteq F$ such that $R \cap C \neq \emptyset$ for each equivalence class $C \in F/\equiv$. The following statement has a straightforward proof using normal forms (see [19, Proposition 7.5] for details).

▶ **Fact 4** ([15]). *Let q and l be fixed non-negative integers. The set $MSO_{q,l}/\equiv$ is finite, and one can compute a system of representatives of $MSO_{q,l}/\equiv$.*

We will assume that for any pair of non-negative integers q and l the system of representatives of $\mathrm{MSO}_{q,l}/\equiv$ given by Fact 4 is fixed.

3 (k,c)-Well-Structured Modulators

▶ **Definition 1.** Let $\mathcal{H}$ be a graph class and let G be a graph. A set $\vec{X}$ of pairwise-disjoint split-modules of G is called a (k,c)-*well-structured modulator* to $\mathcal{H}$ if

1. $|\vec{X}| \leq k$, and
2. $\bigcup_{X_i \in \vec{X}} X_i$ is a modulator to $\mathcal{H}$, and
3. $rw(G[X_i]) \leq c$ for each $X_i \in \vec{X}$.

For the sake of brevity and when clear from context, we will sometimes identify $\vec{X}$ with $\bigcup_{X_i \in \vec{X}} X_i$ (for instance $G - \vec{X}$ is shorthand for $G - \bigcup_{X_i \in \vec{X}} X_i$). To allow a concise description of our parameters, for any hereditary graph class $\mathcal{H}$ we let the *well-structure number* ($wsn_c^{\mathcal{H}}$ in short) denote the minimum k such that G has a (k,c)-well-structured modulator to $\mathcal{H}$.

We conclude this section with a brief discussion on the choice of the parameter. The specific conditions restricting the contents of the modulator $\bigcup \vec{X}$ have been chosen as the most general means which allow both (1) the efficient finding of a suitable well-structured modulator, and (2) the efficient use of this well-structured modulator for kernelization. In this sense, we do not claim that there is anything inherently special about rank-width or split modules, other than being the most general notions which are currently known to allow the achievement of these two goals.

In some of the applications of our results, we will consider graphs which have bounded expansion or bounded degree. We remark that in these cases, our results could equivalently be stated in terms of treewidth (instead of rank-width) and MSO_2 logic (instead of MSO_1 logic).?

4 A Case Study: Vertex Cover

In this section we show how well-structured modulators to edgeless graphs can be used to obtain polynomial kernels for various problems. In particular, this special case can be viewed as a generalization of the vertex cover number. We begin by comparing the resulting parameter to known structural parameters. Let $c \in \mathbb{N}$ be fixed and $\mathcal{E}$ denote the class of edgeless graphs. The class $\mathcal{Z}$ containing only the empty graph will also be of importance later on in the section; we remark that while $mod^{\mathcal{Z}}$ represents a very weak parameter as it is equal to the order of the graph, this is not the case for $wsn_c^{\mathcal{Z}}$. We begin by comparing well-structured modulators to edgeless graphs with similar parameters used in kernelization.

▶ **Proposition 2.** *Let $\mathcal{E}$ be graph class of edgeless graphs. Then:*

1. *$rwc_c(G) \geq wsn_c^{\mathcal{E}}(G)$ for any graph G. Furthermore, for every $i \in \mathbb{N}$ there exists a graph G_i such that $rwc_c(G_i) \geq 2i$ and $wsn_c^{\mathcal{E}} = 2$.*
2. *$vcn(G) \geq wsn_1^{\mathcal{E}}(G)$ for any graph G. Furthermore, for every $i \in \mathbb{N}$ there exists a graph G_i such that $vcn(G) \geq i$ and $wsn_1^{\mathcal{E}} = 1$.*

It will be useful to observe that the above Proposition 2 also holds when restricted to the class of graphs of bounded expansion and bounded degree, and even when the graph class $\mathcal{E}$ is replaced by $\mathcal{Z}$.

As we have established that already $wsn_1^{\mathcal{E}} \leq vcn(G)$, it is important to mention that an additional structural restriction on the graph is necessary to allow the polynomial kernelization of MSO-Opt problems in general (as is made explicit in the following Fact 5).

▶ **Fact 5** ([7]). Clique *parameterized by the vertex cover number does not admit a polynomial kernel, unless* NP $\subseteq$ coNP/poly.

However, it turns out that restricting the inputs to graphs of bounded expansion completely changes the situation: under this condition, it is not only the case that all all MSO-MC and MSO-Opt problems admit a linear kernel when parameterized by the vertex cover number, but also when parameterized by the more general parameter $wsn_c^{\mathcal{E}}$. To prove these claims, we begin by stating the following result.

▶ **Fact 6** ([13]). *Let $\mathcal{K}$ be a graph class with bounded expansion. Suppose that for $G \in \mathcal{K}$ and $S \in V(G)$, $\mathcal{C}_1, \ldots, \mathcal{C}_s$ are sets of connected components of $G - S$ such that for all pairs $C, C' \in \cup_i \mathcal{C}_i$ it holds that $C, C' \in \mathcal{C}_j$ for some j if and only if $N_S(C) = N_S(C')$. Let $\delta \geq 0$ be a constant bound on the diameter of these components, i.e., for all $C \in \cup_i \mathcal{C}_i$, $diam(G[V(C)]) \leq \delta$. Then there can be only at most $\mathcal{O}(|S|)$ such sets $\mathcal{C}_i$.*

This allows us to establish a key link between $wsn_c^{\mathcal{E}}$ and $wsn_c^{\mathcal{Z}}$ on graphs of bounded expansion.

▶ **Lemma 3.** *Let $\mathcal{K}$ be a graph class with bounded expansion. Then there exists a constant d such that for every $G \in \mathcal{K}$ it holds that $wsn_c^{\mathcal{Z}}(G) \leq d \cdot (wsn_c^{\mathcal{E}}(G))$.*

Proof. Let $k = wsn_c^{\mathcal{E}}(G)$ and let $\vec{H}$ be a (k, c)-well-structured modulator to $\mathcal{E}$. Let S be a set of vertices containing exactly one vertex from the frontier of every split-module in $\vec{H}$. The graph $G' = G - (\vec{H} - S)$ is a graph with bounded expansion and S is its vertex cover. Clearly, the diameter of every connected component of $G' \setminus S$ is at most 1 (every connected component is a singleton). Therefore, by Fact 6 there exists a constant d' such that there are at most $d' \cdot |S| = d' \cdot wsn_c^{\mathcal{E}}(G)$ sets of vertices $\mathcal{C}_1, \ldots, \mathcal{C}_s$ in $G' - S$ such that for all pairs $v, v' \in \cup_i \mathcal{C}_i$ it holds that $v, v' \in \mathcal{C}_j$ for some j if and only if $N_S(v) = N_S(v')$. Clearly each such $\mathcal{C}_i$ is a split-module in G', and hence also in G. Furthermore, each such $\mathcal{C}_i$ has rank-width at most 1. Hence $wsn_c^{\mathcal{Z}}(G) \leq wsn_c^{\mathcal{E}}(G) + d' \cdot wsn_c^{\mathcal{E}}(G)$. ◀

The above lemma allows us to shift our attention from modulators to $\mathcal{E}$ to a partition of the vertex set into split-modules of bounded rank-width. The rest of this section is then dedicated to proving our results for well-structured modulators to $\mathcal{Z}$. Our proof strategy for this special case of well-structured modulators closely follows the replacement techniques used to obtain the kernelization results for the rank-width cover number [15], with the distinction that many of the tools and techniques had to be generalized to cover splits instead of modules.

▶ **Theorem 4.** *Let $\mathcal{K}$ be a graph class of bounded expansion, $\mathcal{E}$ be the class of edgeless graphs and $\mathcal{Z}$ be the class of empty graphs. For every MSO sentence φ the problem* MSO-MC$_\varphi$ *admits a linear kernel parameterized by $wsn_c^{\mathcal{Z}}$. Furthermore, the problem* MSO-MC$_\varphi$ *admits a linear kernel parameterized by $wsn_c^{\mathcal{E}}$ on $\mathcal{K}$.*

Sketch of Proof. By Lemma 3 it is sufficient to show that MSO-MC$_\phi$ admits a linear kernel parameterized by $wsn_c^{\mathcal{Z}}$. Let G be a graph, $k = wsn_c^{\mathcal{Z}}(G)$, and q be the nesting depth of quantifiers in ϕ. By Fact 7 (given in the following section) we can find the set $\vec{X}$ of equivalence classes of $\sim_c^G$ in polynomial time. Clearly, the set $\vec{X}$ is a (k, c)-well-structured modulator to the empty graph. We proceed by using replacement techniques to construct an equivalent graph $(G', \vec{X}')$ such that each $X'_i \in \vec{X}'$ has size bounded by a constant. Since $|\vec{X}'| \leq k$ and $\bigcup \vec{X}' = V(G')$, it follows that G' is an instance of MSO-MC$_\varphi$ of size $\mathcal{O}(k)$. ◀

Next, we combine the approaches used in [15] and [10] to handle MSO-Opt$_\varphi^\Diamond$ problems by using our more general parameters. Similarly as in [15], we use a more involved replacement procedure which explicitly keeps track of the original cardinalities of sets and results in an *annotated version* of MSO-Opt$_\varphi^\Diamond$. However, some parts of the framework (in particular the replacement procedure) had to be reworked using the techniques developed in [10], since we now use split-modules instead of simple modules.

▶ **Theorem 5.** *Let $\mathcal{E}$ be a class of edgeless graphs and $\mathcal{Z}$ be the class containing the empty graph. For every MSO formula φ the problem* MSO-Opt$_\varphi^\leqq$ *admits a linear bikernel parameterized by $wsn_c^{\mathcal{E}}$ on any class of graphs of bounded expansion, and a linear bikernel parameterized by $wsn_c^{\mathcal{Z}}$.*

5 Finding (k, c)-Well-Structured Modulators

For the following considerations, we fix c and assume that the graph G has rank-width at least $c + 2$ (this is important for Fact 7). This assumption is sound, since the considered problems can be solved in polynomial time on graphs of bounded rank-width. Recall that given a split-module A in G, we use $\lambda(A)$ to denote the frontier of A. This section will show how to efficiently approximate well-structured modulators to various graph classes; in particular, we give algorithms for the class of forests and then for any graph class which can be characterized by a finite set of forbidden induced subgraphs.

The following Fact 7 linking rank-width and split-modules will be crucial for approximating our well-structured modulators.

▶ **Definition 6.** Let G be a graph and $c \in \mathbb{N}$. We define a relation $\sim_c^G$ on $V(G)$ by letting $v \sim_c^G w$ if and only if there is a split-module M of G with $v, w \in M$ and $rw(G[M]) \leq c$. We drop the superscript from $\sim_c^G$ if the graph G is clear from context.

▶ **Fact 7** ([10]). *Let $c \in \mathbb{N}$ be fixed and G be a graph of rank-width at least $c + 2$. The relation $\sim_c^G$ is an equivalence, and any graph G has its vertex set uniquely partitioned by*

the equivalence classes of $\sim_c$ into inclusion-maximal split-modules of rank-width at most c. Furthermore, for $a, b \in V(G)$ it is possible to test $a \sim_c b$ in $\mathcal{O}(n^3)$ time.

5.1 Finding (k, c)-Well-Structured Modulators to Forests

Our starting point is the following lemma, which shows that long cycles which hit a non-singleton frontier imply the existence of short cycles.

▶ **Lemma 7.** *Let C be a cycle in G such that C intersects at least three distinct equivalence classes of $\sim_c$, one of which has a frontier of cardinality at least 2. Let Z be the set of equivalence classes of $\sim_c$ which intersect C. Then there exists a cycle C' such that the set Z' of equivalence classes it intersects is a subset of Z and has cardinality at most 3.*

We will use the following observation to proceed when Lemma 7 cannot be applied.

▶ **Observation 1.** *Assume that for each equivalence class B of $\sim_c$ it holds that $G[B]$ is acyclic, and that no cycle intersects B if $|\lambda(B)| \geq 2$. Then for every cycle C in G and every vertex $a \in C$, it holds that a is in the frontier of some equivalence class of $\sim_c$.*

Fact 8 below is the last ingredient needed for the algorithm.

▶ **Fact 8** ([3]). Feedback Vertex Set *can be 2-approximated in polynomial time.*

▶ **Theorem 8.** *Let $c \in \mathbb{N}$ and $\mathcal{F}$ be the class of forests. There exists a polynomial algorithm which takes as input a graph G of rank-width at least $c + 2$ and computes a set $\vec{X}$ of split-modules such that $\vec{X}$ is a (k, c)-well-structured modulator to $\mathcal{F}$ and $k \leq 3 \cdot wsn_c^{\mathcal{F}}$.*

Sketch of Proof. The algorithm proceeds in three steps.

1. By deciding $a \sim_c b$ for each pair of vertices in G as per Fact 7, we compute the equivalence classes of $\sim_c$.
2. For each set of up to three equivalence classes $\{A_1, A_2, A_3\}$ of $\sim_c$, we check if $G[A_1 \cup A_2 \cup A_3]$ is acyclic; if it's not, then we add A_1, A_2 and A_3 to $\vec{X}$ and set $G := G - (A_1 \cup A_2 \cup A_3)$.
3. We use Fact 8 to 2-approximate a feedback vertex set S of G in polynomial time; let S' contain every equivalence class of $\sim_c$ which intersects S. We then set $\vec{X} := \vec{X} \cup S'$, and output $\vec{X}$. ◀

5.2 Finding (k, c)-Well-Structured Modulators via Obstructions

Here we will show how to efficiently find a sufficiently small (k, c)-well-structured modulator to any graph class which can be characterized by a finite set of forbidden induced subgraphs. Let us fix a graph class $\mathcal{H}$ characterized by a set $\mathcal{R}$ of forbidden induced subgraphs, and let r be the maximum order of a graph in $\mathcal{R}$. Our first step is to reduce our problem to the classical Hitting Set problem, the definition of which is recalled below.

> d-Hitting Set
> *Instance*: A ground set S and a collection $\mathcal{C}$ of subsets of S, each of cardinality at most d.
> *Notation*: A hitting set is a subset of S which intersects each set in $\mathcal{C}$.
> *Task*: Find a minimum-cardinality hitting set.

Given a graph G (of rank-width at least $c+2$), we construct an instance W_G of r-Hitting Set as follows. The ground set of W contains each equivalence class $A \subseteq V(G)$ of $\sim_c$. For each induced subgraph $R \subseteq G$ isomorphic to an element of $\mathcal{R}$, we add the set C_R of

equivalence classes of $\sim_c$ which intersect R into $\mathcal{C}$. This completes the construction of W_G; we let $hit(W_G)$ denote the cardinality of a solution of W_G.

▶ **Lemma 9.** *For any graph G of rank-width at least $c+2$, the instance W_G is unique and can be constructed in polynomial time. Every hitting set Y in W_G is a $(|Y|, c)$-well-structured modulator to $\mathcal{H}$ in G. Moreover, $wsn_c^{\mathcal{H}} = hit(W_G)$.*

The final ingredient we need for our approximation algorithm is the following result.

▶ **Fact 9** (Folklore). *There exists a polynomial-time algorithm which takes as input an instance W of r-Hitting Set and outputs a hitting set Y of cardinality at most $r \cdot hit(W)$.*

▶ **Theorem 10.** *Let $c \in \mathbb{N}$ and $\mathcal{H}$ be a class of graphs characterized by a finite set of forbidden induced subgraphs of order at most r. There exists a polynomial algorithm which takes as input a graph G of rank-width at least $c+2$ and computes a (k, c)-well-structured modulator to $\mathcal{H}$ such that $k \leq r \cdot wsn_c^{\mathcal{H}}$.*

Proof. We proceed in two steps: first, we compute the r-Hitting Set instance W_G, and then we use Fact 9 to compute an r-approximate solution Y of W_G in polynomial time. We then set $\vec{X} := Y$ and output. Correctness follows from Lemma 9. ◀

6 Applications of (k, c)-Well-Structured Modulators

We now proceed by outlining the general applications of our results. Our algorithmic framework is captured by the following Theorem 11.

▶ **Theorem 11.** *Let p, q be polynomial functions. For every MSO sentence ϕ and every graph class $\mathcal{H}$ such that*

1. MSO-MC$_\phi$ *admits a (bi)kernel of size $p(mod^{\mathcal{H}}(G))$, and*
2. *there exists a polynomial algorithm which finds a $(q(wsn_c^{\mathcal{H}}), c)$-well-structured modulator to $\mathcal{H}$.*

Then MSO-MC$_\phi$ *admits a (bi)kernel of size $p(q(wsn_c^{\mathcal{H}}(G)))$.*

Let us briefly discuss the limitations of the above theorem. The condition that MSO-MC$_\phi$ admits a polynomial (bi)kernel parameterized by $mod^{\mathcal{H}}(G)$ is clearly necessary for the rest of the theorem to hold, since $wsn^{\mathcal{H}}(G) \leq mod^{\mathcal{H}}(G)$. One might wonder whether a weaker necessary condition could be used instead; specifically, would it be sufficient to require that MSO-MC$_\phi$ is polynomial-time tractable in $\mathcal{H}$? This turns out not to be the case, as follows from the following fact.

▶ **Fact 10** ([10]). *There exists an MSO sentence ϕ and a graph class $\mathcal{H}$ characterized by a finite set of forbidden induced subgraphs such that* MSO-MC$_\phi$ *is polynomial-time tractable on $\mathcal{H}$ but NP-hard on the class of graphs with $mod^{\mathcal{H}}(G) \leq 2$.*

Condition 2 is also necessary for our approach to work, as we need some (approximate) well-structured modulator; luckily, Section 5 shows that a wide variety of studied graph classes satisfy this condition. Finally, one can also rule out an extension of Theorem 11 to MSO-Opt problems (which was possible in the special case considered in Section 4), as we show below.

▶ **Lemma 12.** *There exists an MSO formula φ and a graph class $\mathcal{H}$ characterized by a finite obstruction set such that* MSO-Opt$^{\leq}_{\varphi}$ *admits a bikernel parameterized by $mod^{\mathcal{H}}$ but is* paraNP*-hard parameterized by $wsn_1^{\mathcal{H}}$.*

Sketch of Proof. Consider the formula $\varphi(S) = \text{fvs}(S) \vee \deg(S)$, where $\text{fvs}(S)$ expresses that S is a feedback vertex set in G and $\deg(S)$ expresses that S is a modulator to graphs with maximum degree 4. Let $\mathcal{H}$ be the class of graphs of maximum degree 4. ◀

6.1 Applications of Theorem 11

As our first general application, we consider the results of Gajarský et al. in [13]. Their main result is summarized below.

▶ **Fact 11** ([13]). *Let Π be a problem with finite integer index, $\mathcal{K}$ a class of graphs of bounded expansion, $d \in \mathbb{N}$, and $\mathcal{H}$ be the class of graphs of treedepth at most d. Then there exist an algorithm that takes as input $(G, \xi) \in \mathcal{K} \times \mathbb{N}$ and in time $\mathcal{O}(|G| + \log \xi)$ outputs (G', ξ') such that*

1. *$(G, \xi) \in \Pi$ if and only if $(G', \xi') \in \Pi$;*
2. *G' is an induced subgraph of G; and*
3. *$|G'| = \mathcal{O}(mod^{\mathcal{H}}(G))$.*

The following fact provides a link between the notion of *finite integer index* used in the above result and the MSO-MC$_\varphi$ problems considered in this paper.

▶ **Fact 12** ([2], see also [4]). *For every MSO sentence φ, it holds that* MSO-MC$_\varphi$ *is finite-state and hence has finite integer index.*

Finally, the following well-known fact is the last ingredient we need to apply our machinery.

▶ **Fact 13** ([20], page 138). *Let $d \in \mathbb{N}$ and $\mathcal{H}$ be the class of graph of treedepth at most d. Then $\mathcal{H}$ can be characterized by finite set of forbidden induced subgraphs.*

▶ **Theorem 13.** *Let $c, d \in \mathbb{N}$ and $\mathcal{H}$ be the class of graphs of treedepth at most d. For every MSO sentence φ, it holds that* MSO-MC$_\varphi$ *admits a linear kernel parameterized by $wsn_c^{\mathcal{H}}$ on any class of graphs of bounded expansion.*

As our second general application, we consider well-structured modulators to the class of forests. Lemma 14 shows that feedback vertex set may be used to kernelize any MSO-definable decision problem on graphs of bounded degree.

▶ **Lemma 14.** *Let $\mathcal{F}$ be the class of forests and $d \in \mathbb{N}$. For every MSO sentence φ, it holds that* MSO-MC$_\varphi$ *admits a linear kernel parameterized by $mod^{\mathcal{F}}$ on any class of graphs of degree at most d.*

With Lemma 14, the proof of the theorem below is analogous to the proof of Theorem 13.

▶ **Theorem 15.** *Let $c \in \mathbb{N}$ and $\mathcal{F}$ be the class of forests. For every MSO sentence φ, it holds that* MSO-MC$_\varphi$ *admits a linear kernel parameterized by $wsn_c^{\mathcal{F}}$ on any class of graphs of bounded degree.*

7 Conclusion

Our results show that measuring the structure of modulators can lead to an interesting and, as of yet, relatively unexplored spectrum of structural parameters. Such parameters have the potential of combining the best of decomposition-based techniques and modulator-based techniques, and can be applied both in the context of kernelization (as demonstrated in this

work) and FPT algorithms [10]. We believe that further work in the direction of modulators will allow us to push the frontiers of tractability towards new, uncharted classes of inputs.

One possible direction for future research is the question of whether the class of MSO-definable problems considered in Theorem 11 can be extended to other finite-state problems. It would of course also be interesting to see more applications of Theorem 11 and new methods for approximating well-structured modulators. Last but not least, we mention that the split-modules used in the definition of our parameters could in principle be refined to less restrictive notions (for instance cuts of constant *cut-rank* [21]); such a relaxed parameter could still be used to obtain polynomial kernels, as long as there is a way of efficiently approximating or computing such modulators.

References

1 Noga Alon, Gregory Gutin, Eun Jung Kim, Stefan Szeider, and Anders Yeo. Solving max-r-sat above a tight lower bound. *Algorithmica*, 61(3):638–655, 2011.

2 Stefan Arnborg, Bruno Courcelle, Andrzej Proskurowski, and Detlef Seese. An algebraic theory of graph reduction. *J. of the ACM*, 40(5):1134–1164, 1993.

3 Ann Becker and Dan Geiger. Optimization of pearl's method of conditioning and greedy-like approximation algorithms for the vertex feedback set problem. *Artif. Intell.*, 83(1):167–188, 1996.

4 Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (meta) kernelization. In *FOCS 2009*, pages 629–638. IEEE Computer Society, 2009.

5 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernel bounds for path and cycle problems. *Theor. Comput. Sci.*, 511:117–136, 2013.

6 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Preprocessing for treewidth: A combinatorial analysis through kernelization. *SIAM J. Discrete Math.*, 27(4):2108–2142, 2013.

7 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM J. Discrete Math.*, 28(1):277–305, 2014.

8 Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer Verlag, New York, 2nd edition, 2000.

9 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer Verlag, 2013.

10 Eduard Eiben, Robert Ganian, and Stefan Szeider. Solving problems on graphs of high rank-width. In *Algorithms and Data Structures – 14th International Symposium*, volume 9214 of *Lecture Notes in Computer Science*, pages 314–326. Springer, 2015.

11 Fedor V. Fomin, Bart M. P. Jansen, and Michal Pilipczuk. Preprocessing subgraph and minor problems: When does a small vertex cover help? *J. Comput. Syst. Sci.*, 80(2):468–495, 2014.

12 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Bidimensionality and kernels. In *SODA*, pages 503–510, 2010.

13 Jakub Gajarský, Petr Hliněný, Jan Obdržálek, Sebastian Ordyniak, Felix Reidl, Peter Rossmanith, Fernando Sanchez Villaamil, and Somnath Sikdar. Kernelization using structural parameters on sparse graph classes. In *ESA 2013*, volume 8125 of *Lecture Notes in Computer Science*, pages 529–540. Springer, 2013.

14 Robert Ganian and Petr Hliněný. On parse trees and Myhill-Nerode-type tools for handling graphs of bounded rank-width. *Discr. Appl. Math.*, 158(7):851–867, 2010.

15 Robert Ganian, Friedrich Slivovsky, and Stefan Szeider. Meta-kernelization with structural parameters. In *MFCS*, pages 457–468, 2013.

16 Petr Hliněný and Sang il Oum. Finding branch-decompositions and rank-decompositions. *SIAM J. Comput.*, 38(3):1012–1032, 2008.

17 Bart M. P. Jansen and Hans L. Bodlaender. Vertex cover kernelization revisited – upper and lower bounds for a refined parameter. *Theory Comput. Syst.*, 53(2):263–299, 2013.

18 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. In *ICALP (1)*, pages 613–624, 2013.

19 Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004.

20 Jaroslav Nesetril and Patrice Ossona de Mendez. *Sparsity – Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and Combinatorics*. Springer, 2012.

21 Sang-il Oum and P. Seymour. Approximating clique-width and branch-width. *J. Combin. Theory Ser. B*, 96(4):514–528, 2006.

Parameter Compilation

Hubie Chen

University of the Basque Country (UPV/EHU), E-20018 San Sebastián, Spain *and* IKERBASQUE, Basque Foundation for Science, E-48011 Bilbao, Spain

Abstract

In resolving instances of a computational problem, if multiple instances of interest share a feature in common, it may be fruitful to compile this feature into a format that allows for more efficient resolution, even if the compilation is relatively expensive. In this article, we introduce a formal framework for classifying problems according to their compilability. The basic object in our framework is that of a parameterized problem, which here is a language along with a parameterization – a map which provides, for each instance, a so-called parameter on which compilation may be performed. Our framework is positioned within the paradigm of parameterized complexity, and our notions are relatable to established concepts in the theory of parameterized complexity. Indeed, we view our framework as playing a unifying role, integrating together parameterized complexity and compilability theory.

1998 ACM Subject Classification F.1.3 [Computation by Abstract Devices] Complexity measures and classes

Keywords and phrases compilation, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.127

1 Introduction

In resolving instances of a computational problem, if it is the case that multiple instances of interest share a feature in common, it may be fruitful to *compile* this feature into a format that allows for more efficient resolution, even if the compilation is relatively expensive. As a first, simple example, consider the problem of deciding if two nodes of an undirected graph are connected. If it is anticipated that many such connectivity queries will share the same graph G, it may be worthwhile to compile G into a format that will allow for accelerated resolution of the queries. As a second example, consider the problem of evaluating a database query on a database. If one is interested in a small set of queries that will be posed to numerous databases, it may be worthwhile to compile the queries of interest into a format that allows for the fastest evaluation. Note that a relatively expensive compilation process may be worthwhile if its results are amortized by repeated use. Indeed, one may conceive of compilation as an off-line preprocessing, whose expense is offset by its later on-line use.

In this article, we attempt to make an infrastructural contribution by introducing a formal framework for classifying problems according to their compilability. Such a framework was previously presented by Cadoli, Domini, Liberatore, and Schaerf [2], (hereafter, *CDLS*); we will discuss the relationship between our framework and theirs below.

The basic object in our framework is a *paramaterized problem*, which we define to be a language Q along with a *parameterization* κ, a polynomial-time computable mapping defined from strings to strings. (For precise details and justifications of definitions, refer to the technical sections of the article.) As usual, we refer to $\kappa(x)$ as the *parameter* of an instance x. In our framework, we wish to understand for which problems the parameters can be succinctly compiled into a form such that, post-compilation, the problem can be

10th International Symposium on Parameterized and Exact Computation (IPEC 2015).
Editors: Thore Husfeldt and Iyad Kanj; pp. 127–137

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

resolved in polynomial-time. The base class of our framework, called poly-comp-PTIME, is (essentially) defined to contain a parameterized problem (Q, κ) if there exists a polynomial-length, computable function c such that if each instance x is always presented along with $c(\kappa(x))$, then each instance can be resolved in polynomial time. The function c models the notion of compilation of the parameters. In order to give evidence of non-containment in the class poly-comp-PTIME and also to facilitate problem classification, we introduce a hierarchy of parameterized complexity classes chopped-$\mathcal{C}$, one for each classical complexity class $\mathcal{C}$; we observe (for example) that chopped-NP is not contained in poly-comp-PTIME, assuming that the polynomial hierarchy does not collapse (see Proposition 9 and Theorem 19), and hence hardness of a problem for chopped-NP can be construed as evidence of non-containment in poly-comp-PTIME. We observe a number of completeness and hardness results for chopped-NP (Section 6).[1] The class poly-comp-PTIME and the classes chopped-$\mathcal{C}$ are all subsets of the parameterized class FPT, which is considered to be the basic notion of tractability in the paradigm of parameterized complexity.[2] We believe that the introduced classes constitute a natural stratification of FPT, whose study might well lead to deeper theory.

In the CDLS framework, the basic object is a language consisting of pairs of strings (called a *language of pairs*), and one aims to understand when a compilation can be applied to the first entry of each pair so as to allow for efficient decision. This is a point of difference with our framework, but note that the notions from our framework can be readily applied to the languages of pairs that CDLS study by using the parameterization π_1 that returns the first entry of a pair. Another point of difference between our framework and theirs is that their analog of our compilation function c is not required to be computable; while this makes the negative results stronger, in our view there ought to be a focus on positive results, which are rendered less meaningful without the computability requirement. (Actually, we are not aware of any natural computable problem for which the presence or absence of this requirement makes a difference.) Although these differences may appear slight, by initiating our theory with our particular choice of definitions, we are able to position our framework within the language and tradition of parameterized complexity and relate our notions to existing ideas in parameterized complexity. For instance, although not difficult, we can directly relate the notion of a *polynomial kernelization* to the classes chopped-$\mathcal{C}$ (Proposition 10) and use this relationship to observe the chopped-NP-completeness of the standard parameterization of the hitting set problem for hypergraphs of bounded edge size (see Theorem 30). We also believe that the theory that results from our framework's definitions witnesses that working with parameterized problems as opposed to languages of pairs allows for greater flexibility and smoother formulation (consider, for example, the characterization of chopped-$\mathcal{C}$ using the *length parameterization* given by Proposition 14).

Our framework and that of CDLS also differ later in the respective developments. Notably, our notion of reduction (Definition 11) is readily seen to be a restricted version of the usual fpt many-one reduction in parameterized complexity, and we believe that our notion of reduction is conceptually simpler to comprehend than that of CDLS [2, Definition 2.8]. Despite these differences – and we view this as crucial – we demonstrate how classification results obtained in the CDLS framework can be formulated and obtained in our framework; this is made precise and performed in Section 5.

---

[1] These results include a hardness result on model checking existential positive sentences (Proposition 31); we remark that obtaining a broader understanding of the non-compilability results in the author's previous study of model checking [5] was in fact a motivation of the present article.

[2] Note that the containment of poly-comp-PTIME in FPT is essentially observed (in different language) in the last paragraph of Section 5 of [2].

The presentation and development of our framework may thus be viewed as playing a unifying role, integrating together parameterized complexity and compilation. Our choices of definitions and in formulation allows us to directly relate the resulting concepts to the theory of parameterized complexity. At the same time, we believe that these concepts capture in an essential way the core mathematical content and the core ideas of the CDLS framework (as borne out by our results and discussion in Section 5).

Related work. The CDLS framework was deployed after its introduction to analyze the compilability of reasoning tasks, see for example [10, 11].

In the context of compiling propositional formulas, a notion of compilation whereby a compiled version should have the same models as the original formula was studied, for example by Gogic et al. [9] and by Darwiche and Marquis [6]; see also the recent work by Bova et al. [1].

Variants of the CDLS framework that relaxed the requirement that the size of compilations be polynomial were also studied [3, 4].

Finally, we mention that Fan, Geerts, and Neven [7] also developed a framework for classifying problems according to compilability, with a focus on efficient parallel processing (modelled using the complexity class NC) following a polynomial-time compilation. We believe that it may be of interest to better understand and develop the relationship between our framework and theirs. While we leave such a study to future work, we mention that their notion of Π-*tractability* on a language Q of pairs can be described using our framework.[3]

2 Preliminaries

Throughout, π_i denotes the operator that, given a tuple, returns the ith entry of the tuple.

When T is a set, we use $\wp_{\text{fin}}(T)$ to denote the set $\{S \subseteq T \mid S \text{ is finite}\}$.

We generally use Σ to denote the alphabet over which strings are formed, and generally assume $\{0,1\} \subseteq \Sigma$. As is standard, we freely interchange between elements of Σ^* and $\Sigma^* \times \Sigma^*$. When $k \geq 0$, we use $\Sigma^{\leq k}$ to denote the set of strings in Σ^* of length less than or equal to k. For $n \in \mathbb{N}$, we use $\text{un}(n)$ to denote its unary encoding 1^n as a string.

We assume that languages under discussion are non-trivial, that is, not equal to $\emptyset$ nor Σ^*. We use PTIME to denote the set of all languages decidable in polynomial time, and fPTIME to denote the set of all functions from Σ^* to Σ^* that are computable in polynomial time.

Here, by a *parameterization*, we refer to a map from Σ^* to Σ^*. Relative to a parameterization $\kappa : \Sigma^* \to \Sigma^*$, it is typical to refer to $\kappa(x)$ as the *parameter* of the string x. While it is typical in the literature to define a parameterization to be a map from Σ^* to $\mathbb{N}$, in this article we want to apply compilation functions to parameters and discuss the *length* of the results, and we find that this is facilitated in many cases by permitting the parameter of a string to be a string itself. Throughout, we employ the following assumption (which is discussed below in Remark 4).

▶ **Assumption 1.** Each parameterization is polynomial-time computable, that is, in fPTIME.

We use len to denote the parameterization defined by $\text{len}(x) = \text{un}(|x|)$. A *parameterized problem* is a pair (Q, κ) consisting of a language Q and a parameterization κ.

[3] Precisely, a language Q of pairs being Π-*tractable* can be verified to be equivalent to the parameterized problem (Q, π_1) being in our class poly-comp-NC via a poly-compilable function $g(x,y) = f(c(\pi_1(x,y)),(x,y)) = f(c(x),(x,y))$ where c is polynomial-time computable.

By a *classical complexity class*, we refer to a set of computable languages. For a classical complexity class $\mathcal{C}$, we define para-$\mathcal{C}$ to be the set that contains a parameterized problem (Q, κ) if there exists a computable function $c : \Sigma^* \to \Sigma^*$, and a language $Q' \subseteq \Sigma^* \times \Sigma^*$ in $\mathcal{C}$ such that, for each string $x \in \Sigma^*$, it holds that $x \in Q \Leftrightarrow (c(\kappa(x)), x) \in Q'$. We define FPT to be para-PTIME (although this is perhaps not the usual definition of FPT, it is equivalent [8, Theorem 1.37]).

As usual, when $\mathcal{D}$ is a set of problems (that is, a set of either languages or parameterized problems), we say that a problem P' is $\mathcal{D}$-*hard* under a notion of reduction if each P in $\mathcal{D}$ reduces to P'; if in addition $P' \in \mathcal{D}$, we say that P' is $\mathcal{D}$-*complete*. We say that $\mathcal{D}$ is *closed* under a notion of reduction if, when P reduces to P' and $P' \in \mathcal{D}$, it holds that $P \in \mathcal{D}$.

3 Framework

3.1 Problem classes

In this subsection, we introduce the complexity classes of our framework. We begin by introducing two basic definitions. By a *length function*, we refer to a function from $\mathbb{N}$ to $\mathbb{N}$.

▶ **Definition 2.** Let $\mathcal{L}$ be a set of length functions.

- A function $c : \Sigma^* \to \Sigma^*$ is said to be $\mathcal{L}$-length if there exists $\ell \in \mathcal{L}$ such that for each $x \in \Sigma^*$, it holds that $|c(x)| \leq \ell(|x|)$.
- A function $g : \Sigma^* \to \Sigma^*$ is $\mathcal{L}$-*compilable* with respect to a parameterization κ if there exist $f \in$ FPTIME and a computable, $\mathcal{L}$-length function $c : \Sigma^* \to \Sigma^*$ such that (for each $x \in \Sigma^*$) $g(x) = f(c(\kappa(x)), x)$.

Put informally, a function g is $\mathcal{L}$-compilable if, when one has the result of applying c to the parameter of an instance x, the value $g(x)$ can be efficiently computed. The function c can be thought of as performing a precomputation or compilation of the parameter. Here, we do not place any restriction on the computational resources needed to compute c, other than requiring that c is computable. We view the requirement that c be computable as natural in terms of claiming positive results, as we find it hard to argue that a non-computable compilation would actually be usable. We do restrict the length of c according to $\mathcal{L}$; we will be most interested in the case where the length of c is polynomially bounded.

With this terminology in hand, we can now define our first classes of parameterized problems.

▶ **Definition 3.** Let $\mathcal{L}$ be a set of length functions, and let $\mathcal{C}$ be a classical complexity class.

- We say that a parameterized problem (Q, κ) is $\mathcal{L}$-*compilable to* $\mathcal{C}$ if there exists a function $g : \Sigma^* \to \Sigma^*$ that is $\mathcal{L}$-compilable (with respect to κ) and a language $Q' \in \mathcal{C}$ such that (for each $x \in \Sigma^*$) $x \in Q \Leftrightarrow g(x) \in Q'$.
- We define $\mathcal{L}$-comp-$\mathcal{C}$ to be the set that contains each parameterized problem that is $\mathcal{L}$-compilable to $\mathcal{C}$.

When $\mathcal{L}$ is the set of all polynomials on $\mathbb{N}$, we define poly-comp-$\mathcal{C}$ as $\mathcal{L}$-comp-$\mathcal{C}$ and speak, for instance, of *poly-compilability*; similarly, when $\mathcal{L}$ is the set $\cup\{O(2^p) \mid p \text{ is a polynomial}\}$ of *exponential functions*, we define exp-comp-$\mathcal{C}$ as $\mathcal{L}$-comp-$\mathcal{C}$ and speak, for instance, of *exp-compilability*.

▶ **Remark 4.** In this paper, the smallest class that we will consider is poly-comp-PTIME, and we will regard an inclusion result in this class as the most positive result demonstrable on a parameterized problem. Suppose that a parameterized problem (Q, κ) is in poly-comp-PTIME via $g(x) = f(c(\kappa(x)), x)$ and Q'. One way to intuitively interpret this inclusion is as follows.

Suppose that the value $c(k)$ is known for parameter values k in a limited range. Then, for each instance $x \in \Sigma^*$ having parameter value $\kappa(x)$ in that limited range, whether or not $x \in Q$ can be determined efficiently, by applying the efficiently computable function f to $(c(\kappa(x)), x)$ and then by invoking an efficient decision procedure for Q'. Indeed, our intention here is to model the notion of efficient decidability modulo knowledge of c; this is why we put into effect Assumption 1.

We observe the following upper bound on each class $\mathcal{L}$-comp-$\mathcal{C}$, which in particular indicates that poly-comp-PTIME $\subseteq$ FPT.

▶ **Proposition 5.** *Let $\mathcal{L}$ be a set of length functions, and let $\mathcal{C}$ be a classical complexity class that is closed under many-one polynomial-time reduction. It holds that $\mathcal{L}$-comp-$\mathcal{C} \subseteq$ para-$\mathcal{C}$.*

Proof. Suppose that (Q, κ) is $\mathcal{L}$-compilable to $\mathcal{C}$ via $g(x) = f(c(\kappa(x)), x)$ and $Q'' \in \mathcal{C}$, so that $x \in Q \Leftrightarrow g(x) \in Q''$. Define $Q' = \{(a, b) \mid f(a, b) \in Q''\}$. The language Q' is many-one polynomial-time reducible to Q'' via f, so $Q' \in \mathcal{C}$. We have $x \in Q \Leftrightarrow (c(\kappa(x)), x) \in Q'$, implying that $Q \in$ para-$\mathcal{C}$. ◀

We now define a family of complexity classes which will be used to classify parameterized problems in FPT according to their compilability, and in particular to give evidence of non-inclusion in poly-comp-PTIME, via hardness results.

▶ **Definition 6.** For each classical complexity class $\mathcal{C}$, we define chopped-$\mathcal{C}$ to be the set that contains each parameterized problem (Q, κ) that is in poly-comp-$\mathcal{C}$ via a function g for which there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ such that (for each $x \in \Sigma^*$) $|g(x)| \leq p(|\kappa(x)|)$.

For the sake of understanding this definition, let us call the restriction of a language Q' to $Q' \cap \Sigma^{\leq k}$ the *chop having magnitude k* of Q'. Then, intuitively speaking, a problem is in chopped-$\mathcal{C}$ if it is in poly-comp-$\mathcal{C}$ via g and Q' where $g(x)$ accesses only a chop (of Q') having magnitude restricted by a polynomial in the parameter of x. The following proposition is clear from the definition of chopped-$\mathcal{C}$.

▶ **Proposition 7.** *For each classical complexity class $\mathcal{C}$, it holds that*

$$\text{chopped-}\mathcal{C} \subseteq \text{poly-comp-}\mathcal{C}.$$

We also have the following upper bound on chopped-$\mathcal{C}$, which shows that the classes chopped-$\mathcal{C}$ constitute a stratification of the class FPT.

▶ **Proposition 8.** *For each classical complexity class $\mathcal{C}$, it holds that*

$$\text{chopped-}\mathcal{C} \subseteq \text{exp-comp-PTIME},$$

and hence that chopped-$\mathcal{C} \subseteq$ FPT *(by Proposition 5).*

Proof. We prove that chopped-$\mathcal{C} \subseteq$ exp-comp-PTIME. Fix $x_N, x_Y \in \Sigma^*$ and $P \in$ PTIME such that $x_Y \in P$ and $x_N \notin P$. Assume that (Q, κ) is in chopped-$\mathcal{C}$ via $g(x) = f(c(\kappa(x)), x)$, the polynomial p, and $Q' \in \mathcal{C}$. Let $c_1^+ : \Sigma^* \to \Sigma^*$ be the function computed by the algorithm that, given $k \in \Sigma^*$, loops over each string y in $\Sigma^{\leq p(|k|)}$ and, for each such string y, outputs 1 or 0 depending on whether or not $y \in Q'$; thus, $|c_1^+(k)| = |\Sigma^{\leq p(|k|)}|$. Define $c^+(k) = (c_1^+(k), c(k), k)$. Let f^+ be a function computed by a polynomial-time algorithm that, given a string $((d_1, d, k), x)$ where d_1 is a string over $\{0, 1\}$ of length $|\Sigma^{\leq p(|k|)}|$, computes $x' = f(d, x)$, computes the bit b of d_1 corresponding to x' (whenever $|x'| \leq p(|k|)$), and outputs x_Y or x_N depending on whether or not $b = 1$ or $b = 0$. The function $g^+(x) = f^+(c^+(\kappa(x)), x)$ witnesses that (Q, κ) is exp-compilable to PTIME: We have that $x \in Q$ iff $f(c(\kappa(x)), x) \in Q'$ iff $f^+((c_1^+(\kappa(x), c(\kappa(x)), \kappa(x)), x) = x_Y$ iff $f^+(c^+(\kappa(x)), x) \in P$. ◀

We observe that our base class poly-comp-PTIME coincides with the class chopped-PTIME, which is the smallest class that we will consider from the hierarchy of classes chopped-$\mathcal{C}$.

▶ **Proposition 9.** chopped-PTIME = poly-comp-PTIME.

The classes chopped-$\mathcal{C}$ can be directly related to kernelization in the following way. Here, we say that a parameterized problem (Q, κ) has a *polynomial kernelization* if there exists a polynomial-time computable function $K : \Sigma^* \to \Sigma^*$ and a polynomial $p : \mathbb{N} \to \mathbb{N}$ such that (for each $x \in \Sigma^*$) $x \in Q \Leftrightarrow K(x) \in Q$ and $|K(x)| \leq p(|\kappa(x)|)$.

▶ **Proposition 10.** *Suppose that a parameterized problem (Q, κ) has a polynomial kernelization and $\mathcal{C}$ is a classical complexity class such that $Q \in \mathcal{C}$. Then, the problem (Q, κ) is in* chopped-$\mathcal{C}$.

Proof. We have that (Q, κ) in poly-comp-$\mathcal{C}$ via K (the function from the definition of polynomial kernelization), since $x \in Q \Leftrightarrow K(x) \in Q$. Moreover, it holds that there exists a polynomial p such that $|K(x)| \leq p(|\kappa(x)|)$ by the definition of polynomial kernelization. ◀

3.2 Reduction

We now introduce a notion of reduction for comparing the compilability of parameterized problems.

▶ **Definition 11.** We say that a parameterized problem (Q, κ) *poly-comp reduces* to another parameterized problem (Q', κ') if there exists a function $g : \Sigma^* \to \Sigma^*$ that is poly-compilable with respect to κ and a poly-length, computable function $s : \Sigma^* \to \wp_{\mathrm{fin}}(\Sigma^*)$ such that (for each $x \in \Sigma^*$) it holds that $x \in Q \Leftrightarrow g(x) \in Q'$ and that $\kappa'(g(x)) \in s(\kappa(x))$.

The notion of poly-comp reduction can be viewed as a restricted version of fpt many-one reduction. (Consider, for example, the definition given by Flum and Grohe [8, Definition 2.1]; the function g in Definition 11 can be seen to be computable by a fpt-algorithm, and the condition on the function s ensures that their condition (3), when reformulated for parameterizations of the type considered here, holds.)

Note that, in Definition 11, we assume that the set $s(x)$ is given according to a standard representation that lists the strings therein; hence, as a consequence of the assumption that s is poly-length, the size $|s(x)|$ of $s(x)$ is bounded above by a polynomial in $|x|$.

We have the following two basic properties of poly-comp reduction.

▶ **Theorem 12.** *For each classical complexity class $\mathcal{C}$, it holds that* poly-comp-$\mathcal{C}$ *is closed under poly-comp reduction.*

▶ **Theorem 13.** *Poly-comp reducibility is transitive.*

We now give an alternative characterization of chopped-$\mathcal{C}$ in terms of poly-comp reduction.

▶ **Proposition 14.** *Let $\mathcal{C}$ be a classical complexity class. A parameterized problem (Q, κ) is in* chopped-$\mathcal{C}$ *if and only if there exists a language $Q' \in \mathcal{C}$ such that (Q, κ) poly-comp reduces to (Q', len).*

From the just-given characterization of chopped-$\mathcal{C}$, we may infer the following two results.

▶ **Proposition 15.** *For each classical complexity class $\mathcal{C}$, the class* chopped-$\mathcal{C}$ *is closed under poly-comp reduction.*

Proof. This is a consequence of Proposition 14 and Theorem 13. ◀

When discussing a class chopped-$\mathcal{C}$*, we assume by default that* hardness *and* completeness *are with respect to poly-comp reducibility.*

▶ **Proposition 16.** *Let $\mathcal{C}$ be a classical complexity class and assume that Q^+ is $\mathcal{C}$-complete under many-one polynomial-time reduction. Then, the parameterized problem (Q^+, len) is complete for* chopped-$\mathcal{C}$*.*

4 Chopped classes and advice

In this section, we relate the classes chopped-$\mathcal{C}$ to advice-based complexity classes; this will allow us to provide evidence of separation between classes of the form chopped-$\mathcal{C}$.

We first present a known notion from computational complexity theory, the notion of an advice version of a complexity class. For each classical complexity class $\mathcal{C}$, we define $\mathcal{C}/\mathrm{poly}$ to be the set that contains a language Q if and only if there exists a poly-length map $a : \{1\}^* \to \Sigma^*$ and a language $Q' \in \mathcal{C}$ such that, for each $x \in \Sigma^*$, it holds that $x \in Q \Leftrightarrow (a(\mathrm{un}(|x|), x)) \in Q'$.

The following theorem shows that containment of one chopped class in another implies a containment in classical complexity.

▶ **Theorem 17.** *Let $\mathcal{C}$ and $\mathcal{C}'$ be classical complexity classes where $\mathcal{C}'$ is closed under many-one polynomial-time reduction. If* chopped-$\mathcal{C} \subseteq$ chopped-$\mathcal{C}'$*, then $\mathcal{C} \subseteq \mathcal{C}'/\mathrm{poly}$.*

To prove this theorem, we first establish a lemma.

▶ **Lemma 18.** *Let $\mathcal{C}'$ be a classical complexity class that is closed under many-one polynomial-time reduction. If Q is a language such that $(Q, \mathrm{len}) \in$* chopped-$\mathcal{C}'$*, then $Q \in \mathcal{C}'/\mathrm{poly}$.*

Proof of Theorem 17. Suppose that $Q \in \mathcal{C}$. By Proposition 14, it holds that (Q, len) is in chopped-$\mathcal{C}$. By hypothesis, it holds that (Q, len) is in chopped-$\mathcal{C}'$. By Lemma 18, it follows that $Q \in \mathcal{C}'/\mathrm{poly}$. ◀

We use Σ_i^{P} and Π_i^{P} (with $i \geq 0$) to denote the classes of the polynomial hierarchy (PH); recall that $\Sigma_0^{\mathsf{P}} = \Pi_0^{\mathsf{P}} = \mathsf{PTIME}$, $\Sigma_1^{\mathsf{P}} = \mathsf{NP}$, and $\Pi_1^{\mathsf{P}} = \mathsf{co\text{-}NP}$. For each $i \geq 0$, let us say that the classes Σ_i and Π_i are at the *ith level* of the PH. Let us say that a class $\mathcal{C}'$ of the PH is *above* another class $\mathcal{C}$ of the PH if they are equal or if the level j of $\mathcal{C}'$ is strictly greater than the level i of $\mathcal{C}$.

▶ **Theorem 19** (follows from [12])**.** *Suppose that $\mathcal{C}$ and $\mathcal{C}'$ are classes of the PH such that $\mathcal{C}'$ is not above $\mathcal{C}$.*

- *If* chopped-$\mathcal{C} \subseteq$ chopped-$\mathcal{C}'$*, then the PH collapses.*
- *A parameterized problem (Q, κ) that is* chopped-$\mathcal{C}$*-hard is not in* chopped-$\mathcal{C}'$*, unless the PH collapses.*

Proof. For the first claim, it follows from Theorem 17 that $\mathcal{C} \subseteq \mathcal{C}'/\mathrm{poly}$; by [12], it follows that the PH collapses. For the second claim, observe that if (Q, κ) is chopped-$\mathcal{C}$-hard, then $(Q, \kappa) \in$ chopped-$\mathcal{C}'$ implies that chopped-$\mathcal{C} \subseteq$ chopped-$\mathcal{C}'$, by the closure of chopped-$\mathcal{C}'$ under poly-comp reduction (Theorem 12). ◀

5 Relationship to the CDLS framework

In this section, we discuss the relationship between our framework and the CDLS framework. We in particular show that, in a sense that we make precise, the completeness results that they obtain for their problem classes can be formulated and obtained in our framework.

By a *language of pairs*, we refer to a subset of $\Sigma^* \times \Sigma^*$.

The CDLS framework defines, for each classical complexity class, a class which they refer to as the *class of problems non-uniformly compilable to a class C*, and which contains languages of pairs [2, Definition 2.7]. We give the following formulation of this definition.

▶ **Definition 20.** A language $B \subseteq \Sigma^* \times \Sigma^*$ of pairs is in mixed-$\mathcal{C}$ if there exists a poly-length, computable function $f : \Sigma^* \times \Sigma^* \to \Sigma^*$ and a language $B' \in \mathcal{C}$ of pairs such that $(x, y) \in B \Leftrightarrow (f(x, \mathrm{un}(|y|)), y) \in B'$.

Note that our definition is not exactly equivalent to theirs; we require that the function f is computable, while they do not. We do not know of any natural language of pairs for which this makes a difference; assuming computability of f will allow us to more readily relate the defined classes to those of our framework.

To illustrate how classification results on languages obtained in the CDLS framework can be obtained in our framework, we discuss three running examples (studied in [2]):

- Define CI *(clause inference)* to be the set of pairs (x, y) where x is a propositional 3CNF formula, y is a clause, and $x \models y$. We assume here that clauses do not contain repeated literals.
- Define MMC *(minimal model checking)* to be the set of pairs (x, y) where x is a propositional formula and y is a minimal model of x. By *minimal*, we mean with respect to the order $\leq$ where $z \leq z'$ if and only if all variables true under z are also true under z'.
- Define CMI *(clause minimal inference)* to be the set of pairs (x, y) where x is a propositional formula and y is a clause that is satisfied by all minimal models of x.

It is known and straightforward to verify that $\mathrm{CI}, \mathrm{MMC} \in$ co-NP and $\mathrm{CMI} \in \Pi_2^{\mathsf{p}}$. It follows immediately that $\mathrm{CI}, \mathrm{MMC} \in$ mixed-co-NP and $\mathrm{CMI} \in$ mixed-Π_2^{p}.

Let us say that a parameterized problem (Q, κ) has *poly-bounded slices* if there exists a polynomial p such that, for each $x \in Q$, it holds that $|x| \leq p(|\kappa(x)|)$. Each of the three parameterized problems (CI, π_1), (MMC, π_1), and (CMI, π_1) have poly-bounded slices (as is readily verified), and it can consequently be verified that $(\mathrm{CI}, \pi_1), (\mathrm{MMC}, \pi_1) \in$ chopped-co-NP and that $(\mathrm{CMI}, \pi_1) \in$ chopped-Π_2^{p}. It is indeed a general fact that when B is a language of pairs where (B, π_1) has poly-bounded slices, the classes mixed-$\mathcal{C}$ and chopped-$\mathcal{C}$ coincide, as made precise by the following theorem.

▶ **Theorem 21.** *Let $\mathcal{C}$ be a classical complexity class closed under many-one polynomial-time reduction. Let B be a language of pairs such that (B, π_1) has poly-bounded slices. Then, B is in* mixed-$\mathcal{C}$ *if and only if (B, π_1) is in* chopped-$\mathcal{C}$.

We now present a formulation of the notion of reduction used in the CDLS framework (see [2, Definition 2.8]).

▶ **Definition 22.** Let A and B be languages of pairs. A *mixed reduction* from A to B is a triple (f_1, f_2, g) of mappings from $\Sigma^* \times \Sigma^*$ to Σ where f_1 and f_2 are poly-length and computable, and g is polynomial-time computable, such that $(x, y) \in A \Leftrightarrow (f_1(x, \mathrm{un}(|y|)), g(f_2(x, \mathrm{un}(|y|)), y)) \in B$.

In analogy to Definition 20, here we require that the functions f_1 and f_2 are computable.

As a way of showing hardness, CDLS present mixed-reductions from languages of the form $\{\epsilon\} \times Q^+$ where Q^+ is a classical language that is hard. For example, they present the following reductions.

▶ **Theorem 23** (follows from [2, Proof of Theorem 2.10])**.** *There exists a* co-NP*-complete problem Q^+ such that there exists a mixed-reduction from $\{\epsilon\} \times Q^+$ to* CI.

▶ **Theorem 24** (follows from [2, Proof of Theorem 3.2]). *There exists a* Π_2^{p}*-complete problem* Q^+ *such that there exists a mixed-reduction from* $\{\epsilon\} \times Q^+$ *to* CMI.

We now present a general theorem showing that exhibiting a reduction from a language of the form $\{\epsilon\} \times Q^+$ yields a hardness result with respect to the classes chopped-$\mathcal{C}$, made precise as follows.

▶ **Theorem 25.** *Suppose that* A *and* B *are languages of pairs such that there exists a mixed reduction from* A *to* B*, and let* $\mathcal{C}$ *be a classical complexity class. If* $A = \{\epsilon\} \times Q^+$ *where* Q^+ *is* $\mathcal{C}$*-complete, then* (B, π_1) *is* chopped-$\mathcal{C}$*-hard.*

▶ **Corollary 26.** *The problem* (CI, π_1) *is* chopped-co-NP*-hard; the problem* (CMI, π_1) *is* chopped-Π_2^{p}*-hard.*

Proof. Follows from Theorems 23, 24 and 25. ◀

The other way in which CDLS show hardness is by presenting a mixed-reduction from a problem that has poly-bounded slices. For example, they prove the following.

▶ **Theorem 27** (follows from [2, Proof of Theorem 3.1]). *There exists a mixed-reduction from* CI *to* MMC.

We show that this form of reduction can be interpreted as a poly-comp reduction, made precise as follows.

▶ **Theorem 28.** *Suppose that* A *and* B *are languages of pairs such that there exists a mixed reduction from* A *to* B*. If* (A, π_1) *has poly-bounded slices, then* (A, π_1) *poly-comp reduces to* (B, π_1)*.*

▶ **Corollary 29.** *There exists a poly-comp reduction from* (CI, π_1) *to* (MMC, π_1)*, and hence (by Corollary 26) the problem* (MMC, π_1) *is* chopped-co-NP*-hard.*

Proof. Immediate from Theorems 27 and 28. ◀

At this point, we can observe that the non-compilability results that CDLS obtain can be obtained in our framework. For example, consider the following. As we have seen (and as stated in Corollaries 26 and 29), the problems (CI, π_1) and (MMC, π_1) are chopped-co-NP-hard. This implies that these two problems are not in chopped-PTIME, unless the PH collapses, via Theorem 19. We can also obtain the non-compilability results in (essentially) the form stated by CDLS: by invoking Theorem 21, it immediately follows that the problems CI and MMC are not in mixed-PTIME, unless the PH collapses. We want to emphasize here that the hardness proofs can be carried out using the notions and concepts of our framework.

6 Completeness and hardness for chopped-NP

In this section, we present completeness and hardness results for the class chopped-NP.

Define HAM-PATH to be the problem of deciding, given an undirected graph G, whether or not G contains a Hamiltonian path; define the parameterization γ so that $\gamma(G)$ is equal to the number of nodes in G. The problems 3-SAT and CIRCUIT-SAT are defined as usual. In the context of 3-SAT, ν is the parameterization that returns, given a formula ϕ, the number of variables that appear in ϕ. In the context of CIRCUIT-SAT, $\mu + \nu$ is the parameterization that returns, given a circuit C, the sum of the number of non-input gates and the number of input gates of C. For each $d \geq 2$, we consider d-HITTING-SET to be the problem where an

instance is a pair (H, k) consisting of a number $k \geq 1$ and a hypergraph H where each edge has size less than or equal to d, and one is to decide whether or not H has a hitting set of size less than or equal to k. Note that here, all numbers are represented in unary.

▶ **Theorem 30.** *The following problems are* chopped-NP*-complete:*
1. $(\text{HAM-PATH}, \gamma)$
2. $(\text{3-SAT}, \nu)$
3. $(\text{CIRCUIT-SAT}, \mu + \nu)$
4. $(d\text{-HITTING-SET}, \pi_2)$*, for each* $d \geq 2$

As a way of witnessing the utility of the presented framework, let us discuss how one of the non-compilability results from a previous paper [5] on the parameterized complexity of model checking can be formulated within this framework. Here, by a *unary signature*, we mean a signature containing only unary relation symbols. Define unary-EP-MC to be the problem of deciding, given a pair $(\phi, \mathbf{B})$ consisting of an existential positive sentence and a finite relational structure, each over the same unary signature, whether or not ϕ evaluates to true on $\mathbf{B}$ (see the paper [5] for definitions and background).

▶ **Proposition 31.** *The parameterized problem* $(\text{unary-EP-MC}, \pi_1)$ *is* chopped-NP*-hard.*

Proof. Let h be the reduction given in [5], which is a many-one polynomial-time reduction from the CNF satisfiability problem to unary-EP-MC where an instance having n variables and m clauses is mapped to an instance of the form $(S_n^m, \mathbf{B})$, where each S_n^m is a sentence. Let g be the map that, given a 3-SAT formula ϕ, eliminates duplicate clauses from ϕ and then maps the result under h. For a 3-SAT formula ϕ with n variables, it will thus hold that there exists a polynomial $C \in O(n^3)$ such that $\pi_1(g(\phi)) \in \{S_n^0, S_n^1, \ldots, S_n^{C(n)}\}$. If we define $s(n) = \{S_n^0, S_n^1, \ldots, S_n^{C(n)}\}$, we thus have that (g, s) is a poly-comp reduction from $(\text{3-SAT}, \nu)$ to $(\text{unary-EP-MC}, \pi_1)$, which yields the result by Theorem 30. ◀

Acknowledgements. This work was supported by the Spanish project TIN2013-46181-C2-2-R, by the Basque project GIU12/26, and by the Basque grant UFI11/45.

References

1 Simone Bova, Florent Capelli, Stefan Mengel, and Friedrich Slivovsky. Expander cnfs have exponential DNNF size. *CoRR*, abs/1411.1995, 2014.

2 Marco Cadoli, Francesco M. Donini, Paolo Liberatore, and Marco Schaerf. Preprocessing of intractable problems. *Information and Computation*, 176(2):89–120, 2002.

3 Hubie Chen. A theory of average-case compilability in knowledge representation. In *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9–15, 2003*, pages 455–460, 2003.

4 Hubie Chen. Parameterized compilability. In *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30-August 5, 2005*, pages 412–417, 2005.

5 Hubie Chen. On the complexity of existential positive queries. *ACM Trans. Comput. Log.*, 15(1), 2014.

6 Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *J. Artif. Intell. Res. (JAIR)*, 17:229–264, 2002.

7 Wenfei Fan, Floris Geerts, and Frank Neven. Making queries tractable on big data with preprocessing. *PVLDB*, 6(9):685–696, 2013.

8 J. Flum and M. Grohe. *Parameterized Complexity Theory.* Springer, 2006.

9 Goran Gogic, Henry A. Kautz, Christos H. Papadimitriou, and Bart Selman. The comparative linguistics of knowledge representation. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada*, pages 862–869, 1995.

10 Paolo Liberatore. The size of MDP factored policies. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July 28 – August 1, 2002, Edmonton, Alberta, Canada.*, pages 267–272, 2002.

11 Paolo Liberatore and Marco Schaerf. Compilability of propositional abduction. *ACM Trans. Comput. Log.*, 8(1), 2007.

12 Chee-Keng Yap. Some consequences of non-uniform conditions on uniform classes. *Theor. Comput. Sci.*, 26:287–300, 1983.

An FPT Algorithm and a Polynomial Kernel for Linear Rankwidth-1 Vertex Deletion

Mamadou Moustapha Kanté[1], Eun Jung Kim[2], O-joung Kwon[3], and Christophe Paul[4]

1 **LIMOS, CNRS – Clermont Université, Université Blaise Pascal, France**
mamadou.kante@isima.fr
2 **LAMSADE, CNRS – Université Paris Dauphine, France**
eunjungkim78@gmail.com
3 **Institute for Computer Science and Control, Hungarian Academy of Sciences, Hungary***
ojoungkwon@gmail.com
4 **LIRMM, CNRS – Université Montpellier, France†**
christophe.paul@lirmm.fr

Abstract

Linear rankwidth is a linearized variant of rankwidth, introduced by Oum and Seymour [Approximating clique-width and branch-width. *J. Combin. Theory Ser. B*, 96(4):514–528, 2006.], and it is similar to pathwidth, which is the linearized variant of treewidth. Motivated from the results on graph modification problems into graphs of bounded treewidth or pathwidth, we investigate a graph modification problem into the class of graphs having linear rankwidth at most one, called the Linear Rankwidth-1 Vertex Deletion (shortly, LRW1-Vertex Deletion). In this problem, given an n-vertex graph G and a positive integer k, we want to decide whether there is a set of at most k vertices whose removal turns G into a graph of linear rankwidth at most one and if one exists, find such a vertex set. While the meta-theorem of Courcelle, Makowsky, and Rotics implies that LRW1-Vertex Deletion can be solved in time $f(k) \cdot n^3$ for some function f, it is not clear whether this problem allows a runtime with a modest exponential function. We establish that LRW1-Vertex Deletion can be solved in time $8^k \cdot n^{\mathcal{O}(1)}$. The major obstacle to this end is how to handle a long induced cycle as an obstruction. To fix this issue, we define the *necklace graphs* and investigate their structural properties. We also show that the LRW1-Vertex Deletion has a polynomial kernel.

1998 ACM Subject Classification F.2.2 Computation on Discrete Structures, G.2.1 Combinatorics Algorithms, G.2.2 Graph Agorithms

Keywords and phrases (linear) rankwidth, distance-hereditary graphs, thread graphs, parameterized complexity, kernelization

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.138

1 Introduction

In a parameterized problem, we are given an instance (x, k), where k is a secondary measurement, called as the *parameter*. The central question in parameterized complexity is whether a parameterized problem admits an algorithm with runtime $f(k) \cdot |x|^{\mathcal{O}(1)}$, equivalently an *FPT*

* Supported by ERC Starting Grant PARAMTIGHT (No. 280152).
† Supported by the "Chercheur d'avenir – Languedoc-Roussillon" project KERNEL

10th International Symposium on Parameterized and Exact Computation (IPEC 2015).
Editors: Thore Husfeldt and Iyad Kanj; pp. 138–150

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

algorithm, where f is a function depending on the parameter k alone, and $|x|$ is the input size. As we study a parameterized problem when its unparameterized decision version is NP-hard, the function f is super-polynomial in general. A parameterized problem admitting such an algorithm is said to be *fixed-parameter tractable*, or *FPT* in short. For many natural parameterized problems, the function f is overwhelming [15] or even non-explicit [23], especially when the algorithm is indicated by a meta-theorem. Therefore, a lot of research effort focus on designing an FPT algorithm with affordable super-exponential part in the runtime. We are especially interested in solving a parameterized problem in *single-exponential* time, that is, in time $c^k \cdot n^{O(1)}$ for some constant c.

A powerful technique to handle parameterized problems is the *kernelization algorithm*. A kernelization algorithm takes an instance (x, k) and outputs an instance (x', k') in time polynomial in $|x| + k$ satisfying that (1) (x, k) is a YES-instance if and only if (x', k') is a YES-instance, (2) $k' \leq k$, and $|x'| \leq g(k)$ for some function g. The reduced instance is called a *kernel* and the function g is called the *size* of the kernel. A parameterized problem is said to admit a *polynomial kernel* if there is a kernelization algorithm that reduces the input instance into an instance with size bounded by a polynomial function $g(k)$ in k.

Many natural graph problems can be expressed as a graph modification problem. Generally, given an input graph G and a fixed set O of elementary operations and a graph property Π, the objective is to transform G into a graph $H \in \Pi$ by applying at most k operations from O. Vertex deletion, edge deletion/addition or contraction are examples of such elementary operations.

The graph property Π having treewidth or pathwidth at most w has received in-depth attention as many problems become tractable on graphs of small treewidth. The celebrated Courcelle's theorem [8] implies that every graph property expressible in monadic second order logic of the second type (MSO_2) can be verified in time $f(w) \cdot n$, when the input n-vertex graph has treewidth at most w. Furthermore, having small treewidth frequently facilitates the design of a dynamic programming algorithm whose runtime is much faster than that of the all-round algorithm from the Courcelle's meta-theorem. Therefore, it is reasonable to measure how close an instance is from "an island of tractability within an ocean of intractable problems" [18].

In the context of treewidth, the deletion problems for the special cases of $w = 0$ and $w = 1$ correspond to the well-known VERTEX COVER and FEEDBACK VERTEX SET problems respectively. More generally, for any fixed w, the corresponding graph modification problem TREEWIDTH-w VERTEX DELETION can be solved in time $f(w, k) \cdot n$ implied by Graph Minor Theory [23] and Courcelle's meta-theorem [8]. As the function f subsumed in the meta theorem is gigantic, it is natural to ask whether the exponential function in the runtime can be rendered realistic. Recent endeavor pursuing this question culminated in establishing that for any fixed w, the TREEWIDTH-w VERTEX DELETION is single-exponential fixed parameter tractable with the deletion number k as the parameter [13, 19].

As for pathwidth, PATHWIDTH-1 VERTEX DELETION was first studied by Philip, Raman, Villanger [22], and later Cygan, Pilipczuk, Pilipczuk, Wojtaszczyk [12] showed that PATHWIDTH-1 VERTEX DELETION can be solved in time $4.65^k \cdot n^{\mathcal{O}(1)}$ and it admits a quadratic kernel. Using the general method developed for TREEWIDTH-w VERTEX DELETION [13, 19], the PATHWIDTH-w VERTEX DELETION problem admits a single exponential FPT algorithm.

Linear rankwidth. *Rankwidth* was introduced by Oum and Seymour [21] for efficiently approximating *clique-width*. *Linear rankwidth* is a linearized variation of rankwidth like

pathwidth is the linearized variant of treewidth. While treewidth and pathwidth are small only on sparse graphs, dense graphs may have small rankwidth or linear rankwidth. For instance, complete graphs, complete bipartite graphs, and threshold graphs [7] have linear rankwidth at most one even though all of them have unbounded treewidth. Rankwidth and linear rankwidth have been intensively studied to generalize the known results for treewidth and pathwidth [2, 9, 17, 20, 21].

Ganian [16] pointed out that some NP-hard problems, such as computing pathwidth, can be solved in polynomial time on graphs of linear rankwidth at most 1. Generally, the meta-theorem by Courcelle, Makowsky, Rotics [10] states that for every graph property Π expressible in monadic second order logic of the first type (MSO_1) and fixed k, there is a cubic-time algorithm for testing whether a graph of rankwidth at most k has property Π. As rankwidth is always less than or equal to linear rankwidth, those problems are tractable on graphs of bounded linear rankwidth as well.

In the same context, it is natural to ask whether there is an FPT algorithm for (Linear) Rankwidth-w Vertex Deletion, that is, a problem asking whether for a given graph G and a positive integer k, G contains a vertex subset of size at most k whose deletion makes G a graph of (linear) rankwidth at most w. It is only known that for fixed w, both problems are FPT from the meta-theorem on graphs of bounded rankwidth [10] and the fact that one vertex deletion can decrease rankwidth or linear rankwidth by at most one. We discuss it in more detail in the last section. However, as the function of k obtained from the meta-theorem is enormous, it is interesting to know whether there is a single-exponential FPT algorithm for both problems, like Treewidth-w Vertex Deletion. Also, to the best of our knowledge, there was no known previous result whether (Linear) Rankwidth-w Vertex Deletion admits a polynomial kernel for any integer w.

Our contributions. In this paper, we show that the Linear Rankwidth-1 Vertex Deletion problem admits a single-exponential FPT algorithm and a polynomial kernel. This is a first step towards a goal of investigating whether (Linear) Rankwidth-w Vertex Deletion is single-exponential FPT or has a polynomial kernel.

Linear Rankwidth-1 Vertex Deletion (LRW1-Vertex Deletion)
Input : A graph G, a positive integer k
Parameter : k
Question : Does G have a vertex subset S of size at most k whose removal makes G a graph of linear rankwidth at most one?

▶ **Theorem 1.1.** *For fixed k and a given graph G with n vertices, the* LRW1-Vertex Deletion *problem can be solved in $8^k \cdot n^{\mathcal{O}(1)}$ time.*

▶ **Theorem 1.2.** *The* LRW1-Vertex Deletion *problem has a polynomial kernel.*

We note that several graph classes with a certain path-like structure have been studied for parameterized vertex deleting problems. Such classes include graphs of pathwidth 1, proper interval graphs [14, 25], unit interval graphs [24, 5], and interval graphs [6]. A common approach in the previous work is to use the characterization of the structures obtained after removing small obstructions. We also characterize graphs excluding small obstructions for graphs of linear rankwidth at most 1 to develop an FPT algorithm and a polynomial kernel.

The main ingredient is to investigate a new class of graphs, called *necklace graphs*, which are close to graphs of linear rankwidth at most 1. To define this class, we use the induced subgraph obstructions for graphs of linear rankwidth at most 1, which are listed in Figures 1

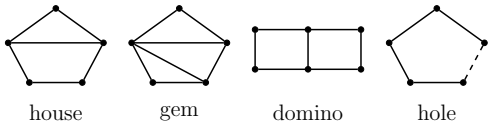


Figure 1 The induced subgraph obstructions for distance-hereditary graphs.

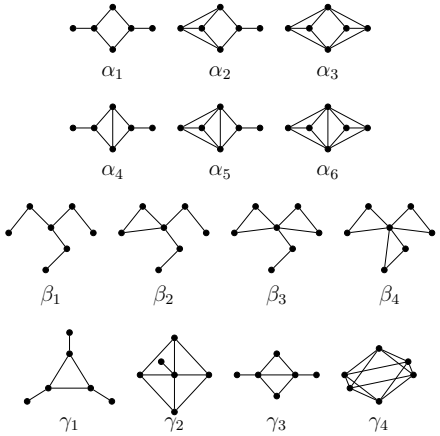


Figure 2 The distance-hereditary induced subgraph obstructions for thread graphs.

and 2 [1]. Briefly speaking, necklace graphs, when viewed locally, are graphs of linear rankwidth at most 1, but they may have a long induced cycle. We show that every connected graph having no obstructions of size at most 8 is a necklace graph, and we can easily find a minimum deleting set on a connected necklace graph. In our FPT algorithm, we first use a simple branching algorithm to remove the obstructions of size at most 8 with the time complexity $8^k \cdot n^{\mathcal{O}(1)}$. Since a final instance does not have obstructions of size at most 8, it is a disjoint union of thread graph and necklace graphs, and we compare the remaining budget with the sum of minimum deleting set over all necklace components to decide whether it is a Yes-instance.

To obtain a polynomial kernel, we start with packing obstructions of size at most 8 using the Sunflower lemma, and taking a minimum deleting set on the remaining necklace graph. The union of two sets will have size bounded by a polynomial in k, and its removal makes an input graph into a graph of linear rankwidth at most 1. Graphs of linear rankwidth at most 1 can be seen as graphs obtained by connecting certain blocks, called *thread blocks*, like a path (Theorem 2.1). The main difficulty for reducing the remaining part is to shrink a large thread block. Even though there is a simple pattern of constructions on thread blocks, it is not at all obvious how to find an irrelevant vertex in a sufficiently large thread block regarding all individual small obstructions. We can resolve this issue using the set obtained by the Sunflower lemma, which has the nice property that any minimum deleting set for the small obstructions in G is contained in the prescribed set. Using this, we will show how to find another obstruction from the long induced cycle containing a potential irrelevant vertex. We remark that a similar idea was used by Fomin, Saurabh, and Villanger [14] to obtain a polynomial kernel for the Proper Interval Vertex Deletion problem.

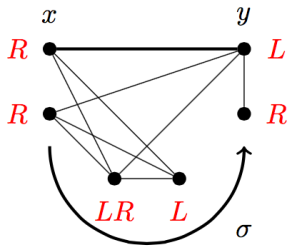


Figure 3 An example of a thread block.

2 Preliminaries

In this paper, all graphs are finite and undirected, if not mentioned otherwise. For a graph G, we denote by $V(G)$ and $E(G)$ the vertex set and the edge set of G, respectively, if they are not specified. Let $G = (V, E)$ be a graph. Let $N_G(x)$ denote the neighborhood of a vertex $x \in V$. For $S \subseteq V$, $G[S]$ denotes the subgraph of G induced on S and we denote by $G \setminus S := G[V \setminus S]$. For $W \subseteq E$, we denote by $G \setminus W := (V, E \setminus W)$. For short we write $G \setminus x$ instead of $G \setminus \{x\}$ for $x \in V \cup E$. A vertex v of G is called a *pendant vertex* if $|N_G(x)| = 1$. An edge e of a connected graph G is called a *cut-edge* if $G \setminus e$ is disconnected. The length of a path is defined as the number of edges in the path. For $n \geq 3$, we denote by C_n the cycle with n vertices. For a set $\mathcal{F}$ of graphs, a graph G is *$\mathcal{F}$-free* if G has no induced subgraph isomorphic to a graph in $\mathcal{F}$.

A (linear) ordering on a finite set S is a bijective mapping $\sigma : S \to \{1, \ldots, |S|\}$, and we write $x <_\sigma y$ if $\sigma(x) < \sigma(y)$, and σ^{-1} as the inverse bijective mapping. For an $X \times Y$-matrix M and $X' \subseteq X, Y' \subseteq Y$, let $M[X', Y']$ be the submatrix of M whose rows and columns are indexed by X' and Y', respectively.

Linear rankwidth and thread graphs. The *adjacency matrix* of a graph $G = (V, E)$, which is a $(0, 1)$-matrix over the binary field, will be denoted by A_G. The *width* of a linear ordering σ of V in G is $\max_{1 \leq i \leq |V|} \operatorname{rank}(A_G[\{\sigma^{-1}(1), \ldots, \sigma^{-1}(i)\}, V \setminus \{\sigma^{-1}(1), \ldots, \sigma^{-1}(i)\}])$, where the rank is computed over the binary field. The *linear rankwidth* of a graph G is defined as the minimum width over all linear orderings of V.

Graphs of linear rankwidth at most one are called *thread graphs* by Ganian [16], and each connected thread graph consists of a sequence of thread blocks. We follow the definition of thread blocks given by Adler, Farley, and Proskurowski [1], and develop a more convenient way to create a thread graph, which is useful to define *necklace graphs*.

A triple $B(x, y) = (G, \sigma, \ell)$, where x and y are two vertices of the graph $G = (V, E)$, σ is a ordering on V, and ℓ is a function from V to $\{\{L\}, \{R\}, \{L, R\}\}$, is a *thread block* if:

1. $\ell(x) = \{R\}$ and $\ell(y) = \{L\}$,
2. for $v, w \in V$ with $v <_\sigma w$, $vw \in E(G)$ if and only if $R \in \ell(v)$ and $L \in \ell(w)$,
3. $\ell(\sigma^{-1}(2)) \neq \{L\}$ if $\sigma^{-1}(2) \neq y$.

See Figure 3 for an example. The aim of the third condition is to guarantee a unique decomposition of thread graphs into thread blocks. For a digraph $D = (V_D, A_D)$, a set of thread blocks $\{B(x, y) = (G_{xy}, \sigma_{xy}, \ell_{xy}) \mid xy \in A_D\}$ is said to be *mergeable with* D if for any two arcs x_1y_1, x_2y_2 of A_D, $V(G_{x_1y_1}) \cap V(G_{x_2y_2}) = \{x_1, y_1\} \cap \{x_2, y_2\}$. For a digraph $D = (V_D, A_D)$ and a mergeable set of thread blocks $\mathcal{B}_D = \{B(x, y) = (G_{xy}, \sigma_{xy}, \ell_{xy}) \mid xy \in$

$A_D\}$, $G = D \odot \mathcal{B}_D$ is the graph with the vertex set $V = \bigcup_{xy \in A_D} V(G_{xy})$ and the edge set $E = \bigcup_{xy \in A_D} E(G_{xy})$.

A connected graph G is a *thread graph* if G is an one vertex graph or $G = P \odot \mathcal{B}_P$ for some directed path P, called the *underlying path*, and some set of thread blocks $\mathcal{B}_P$ mergeable with P. A graph is a *thread graph* if each of its connected components is a thread graph.

The induced subgraph obstructions for graphs of linear rankwidth at most 1 consist of the set of induced subgraph obstructions for graphs of rankwidth at most 1 (equivalently, distance-hereditary graphs) [3], which are a house, a gem, a domino, and induced cycles of length at least 5 in Figure 1, and the set of 14 induced subgraph obstructions for linear rankwidth at most 1 that are graphs of rankwidth 1, depicted in Figure 2. For convenience, we define that

- Ω_U is the set of 14 graphs in Figure 2,
- $\Omega_T := \{\text{house, gem, domino}\} \cup \{C_k \mid k \geq 5\} \cup \Omega_U$, and
- $\Omega_N := \{\text{house, gem, domino}, C_5, C_6, C_7, C_8\} \cup \Omega_U$.

▶ **Theorem 2.1** ([16, 1]). *Let G be a graph. The following are equivalent.*

- *G has linear rankwidth at most 1.*
- *G is a thread graph.*
- *G has no induced subgraph isomorphic to a graph in Ω_T.*

It is known that one can recognize a graph of linear rankwidth 1 in polynomial time using split decompositions of graphs [4, 2], and easily decompose a connected thread graph into thread blocks.

▶ **Theorem 2.2** ([4, 2]). *Let G be a graph on with n vertices and m edges. Then in time $\mathcal{O}(n+m)$, we can test whether G is a thread graph, and if G is a thread graph, then we can decompose each connected component into a sequence of thread blocks in the same time.*

In the remaining part, we frequently use the term 'thread graphs' for graphs of linear rankwidth at most 1. For a graph G and $S \subseteq V$, S is called a *LRW1-deletion set* if $G \setminus S$ is a thread graph.

Necklace graphs. We generalize the construction of thread graphs from directed paths to directed cycles. A connected graph G is called a *necklace graph* if $G = C \odot \mathcal{B}_C$ for some directed cycle C, called the *underlying cycle*, and some set of thread blocks $\mathcal{B}_C$ mergeable with C. Our FPT algorithm and the construction of a polynomial kernel relies on the following characterization of Ω_N-free graphs.

▶ **Theorem 2.3.** *A connected Ω_N-free graph is either a connected thread graph or a necklace graph whose underlying cycle has length at least 9.*

Let us sketch the proof of Theorem 2.3, which constructs the underlying cycle and a set of thread blocks as follows. Let $G = (V, E)$ be a connected Ω_N-free graph and suppose that G is not a thread graph. Since G is Ω_N-free and it is not a thread graph, by Theorem 2.1, G has an induced subgraph isomorphic to C_k for some $k \geq 9$. We prove by induction on $|V|$ that if C is a shortest cycle among induced cycles of length at least 9 in G, then G is a necklace graph whose underlying cycle is C. Let $C := v_1 v_2 \cdots v_k v_1$ be a shortest cycle among induced cycles of length at least 9 in G and for convenience, let $v_{k+1} := v_1$ and $v_{k+2} := v_2$. We regard C as a directed cycle where for each $1 \leq j \leq k$, $v_j v_{j+1}$ is an arc.

If $G = C$, then we are done because C itself is a necklace graph with the underlying cycle C. We may assume that $|V| > |V(C)|$, and choose a vertex $v \in V \setminus V(C)$ such that $G \setminus v$ is

connected. Clearly, $G \setminus v$ is again Ω_N-free graph, and C is a shortest cycle among induced cycles of length at least 9 in $G \setminus v$. By the induction hypothesis, there exists some set of thread blocks $\mathcal{B}_C$ mergeable with C such that $G \setminus v = C \odot \mathcal{B}_C$. The rest of the proof consists in showing that $G = C \odot \mathcal{B}'_C$ for some set of thread blocks $\mathcal{B}'_C$ mergeable with C.

3 An FPT algorithm for LRW1-Vertex Deletion

Our FPT algorithm is a branching algorithm that reduces an input instance to a Ω_N-free graph. As each graph of Ω_N has size at most 8, the announced complexity follows. It remains to prove that given a Ω_N-free graph, a minimum vertex deletion set for LRW1-Vertex Deletion can be found in polynomial time. In fact, we prove that such a set has size at most one per necklace component and identifying such a vertex requires polynomial time.

Using the following proposition, we can find a minimum LRW1-deletion set of a Ω_N-free graph in polynomial time.

▶ **Proposition 3.1.** *Let G be a Ω_N-free graph with n vertices and m edges. We can compute the minimum size of a LRW1-deletion set of G in time $\mathcal{O}(n+m)$, and find such a set S in the same time.*

To prove it, we use the following lemma.

▶ **Lemma 3.2.** *Let G be a connected necklace graph with the underlying cycle C of length at least 4. For each $v \in V(C)$, $G \setminus v$ is a thread graph.*

Proof of Proposition 3.1. Let k be the minimum size of a LRW1-deletion set of G. We remark that each connected component of G is either a thread graph or a necklace graph by Theorem 2.3. For each component H of G, we test whether H is a thread graph or not in time $\mathcal{O}(|V(H)|+|E(H)|)$ using Theorem 2.2. Note that we should remove at least one vertex for each necklace component of G, and moreover, by Lemma 3.2, it is enough to remove exactly one vertex for each component. Thus, k is the number of its necklace components.

To identify such a vertex in each necklace component, it is sufficient to find a vertex on the underlying cycle by Lemma 3.2. Let H be a necklace component and C be the underlying cycle of H. Observe that for $v \in V(H) \setminus V(C)$, $H \setminus v$ is still a connected necklace graph with the same underlying cycle, because we can adjust the ordering of the thread block containing v into an ordering without v with the restricted labeling. We first test whether H has a cut vertex, and if it has a cut vertex w, then $w \in V(C)$. Otherwise, we search a vertex cut of size 2. Since every necklace graph whose underlying cycle has length at least 4 contains a vertex cut of size 2, we can find it, say $\{v, w\}$. Then one of the two vertices should be contained in C. We test whether $H \setminus v$ or $H \setminus w$ is a thread graph or not. If $H \setminus v$ is a thread graph, then $v \in V(C)$, and otherwise, $w \in V(C)$. Since finding a cut vertex or a vertex cut of size two can be done in linear time, we are done with the time complexity. ◀

Proof of Theorem 1.1. Let (G, k) be an instance of the LRW1-Vertex Deletion problem. First find an induced subgraph of G isomorphic to a graph in Ω_N and branch by removing one of the vertices in the subgraph. Because the maximum size of graphs in Ω_N is 8, we can find such a vertex subset in time $\mathcal{O}(n^8)$ if exists. After the branching algorithm, we transform the given instance (G, k) into at most 8^k sub-instances (G', k') such that each sub-instance consists of a Ω_N-free graph G' and a remaining budget k'. Clearly, (G, k) is a Yes-instance if and only if one of sub-instances (G', k') is a Yes-instance.

Let (G', k') be a sub-instance obtained from the branching algorithm. Since G' is Ω_N-free, by Theorem 2.3, each connected component of G' is either a thread graph or a necklace

graph with the underlying cycle of length at least 9. By Proposition 3.1, we can compute a minimum LRW1-deletion set of G' in time $\mathcal{O}(|V(G'|+|E(G')|)$, and decide whether (G', k') is a YES-instance. By checking all sub-instances, we can decide whether (G, k) is a YES-instance in time $8^k \cdot \mathcal{O}(n+m)$ where m is the number of edges of G. We conclude that the LRW1-VERTEX DELETION problem can be solved in time $8^k \cdot \mathcal{O}(n^8)$. ◀

4 A polynomial kernel for LRW1-Vertex Deletion

We use the Sunflower lemma for packing obstructions of small size. It consists in finding a subset T of the input graph $G = (V, E)$ whose removal turns G into a thread graph with the property that for every set $S \subseteq V$ of size at most k, the following are equivalent (Lemma 4.2):

- S is a minimal vertex set such that $G \setminus S$ has no obstructions in Ω_N.
- S is a minimal vertex set such that $G[T] \setminus S$ has no obstructions in Ω_N.

From this property, if we choose a minimal LRW1-deletion set S in the input graph, then the vertices of $S \setminus T$ should be used to remove at least one long induced cycle. This property is essentially used to find an irrelevant vertex in a large thread block. Moreover, with this set, we can preprocess the instance so that there is no obstruction containing exactly one vertex of T. This would be used to bound the length of the sequence of thread blocks in each connected component, and the number of non-trivial components.

Let $(G = (V, E), k)$ be an instance of LRW1-VERTEX DELETION. We start with an easy reduction rule.

▶ **Reduction Rule 1.** *If G has a component that is a thread graph, then we remove it from G.*

4.1 Packing small obstructions

Let $\mathcal{F}$ be a family of subsets over a set U. A subset $U' \subseteq U$ is called *a hitting set* of $\mathcal{F}$ if for every set $F \in \mathcal{F}$, $F \cap U' \neq \emptyset$. For a graph G and a family of graphs $\mathcal{F}$, a set $S \subseteq V(G)$ is also called *a hitting set* for $\mathcal{F}$ if for every induced subgraph H of G that is isomorphic to a graph in $\mathcal{F}$, $V(H) \cap S \neq \emptyset$. The following lemma can be obtained from the Sunflower lemma.

▶ **Lemma 4.1** ([14]). *Let $\mathcal{F}$ be a family of sets of size at most d over a set U, and let k be a positive integer. Then in time $\mathcal{O}(|\mathcal{F}|(k+|\mathcal{F}|))$, we can find a nonempty set $\mathcal{F}' \subseteq \mathcal{F}$ such that*

1. *for every $U' \subseteq U$ of size at most k, U' is a minimal hitting set of $\mathcal{F}$ if and only if U' is a minimal hitting set of $\mathcal{F}'$, and*
2. *$|\mathcal{F}'| \leq d!(k+1)^d$.*

Using Proposition 3.1 and Lemma 4.1, we identify a subset T of vertices of G of polynomial size in k that allows us to forget about small obstructions in G.

▶ **Lemma 4.2.** *Let $(G = (V, E), k)$ be an instance of* LRW1-VERTEX DELETION. *There is a polynomial time algorithm that either concludes that (G, k) is a* NO*-instance or finds a non-empty set $T \subseteq V$ such that*

1. *$G \setminus T$ is a thread graph,*
2. *for every set $S \subseteq V$ of size at most k, S is a minimal hitting set for Ω_N in G if and only if it is a minimal hitting set for Ω_N contained in $G[T]$, and*
3. *$|T| \leq 8 \cdot 8!(k+1)^8 + k$.*

Let us fix a subset T of V obtained by Lemma 4.2. We preprocess using the following reduction rule.

▶ **Reduction Rule 2.** *Let $U \subseteq T$ such that for every $u \in U$, there exists an induced subgraph H of G isomorphic to a graph in Ω_N with $V(H) \cap T = \{u\}$. If $|U| > k$, then (G, k) is a* No*-instance; otherwise, remove U from G and reduce k by $|U|$, and use $T \setminus U$ instead of T.*

It can be done in polynomial time because we only need to look at obstructions of Ω_N in G.

▶ **Lemma 4.3.** *Reduction Rule 2 is safe.*

From now on, we assume that G is reduced under Reduction Rules 1 and 2. A vertex v of G is called *irrelevant* if (G, k) is a Yes-instance if and only if $(G \setminus v, k)$ is a Yes-instance.

4.2 Bounding the size of components of $G \setminus T$

The goal is to shrink $G \setminus T$ while preserving the solutions. For convenience, let $\mu(k) := 8 \cdot 8!(k+1)^8 + k$. We first show that if a thread block in $G \setminus T$ is large, then we can always find an irrelevant vertex in there.

▶ **Proposition 4.4.** *If $G \setminus T$ contains a thread block $(G_{xy}, \sigma_{xy}, \ell_{xy})$ of size at least $(k+2)(\mu(k)+2)^2+1$, then we can find an irrelevant vertex in G_{xy} in polynomial time.*

We mainly use the following lemma.

▶ **Lemma 4.5.** *Let G be a graph and let $v_1v_2v_3v_4v_5$ be an induced path of length 4 in G. If two distinct vertices w_1, w_2 in $V(G) \setminus \{v_1, v_2, \ldots, v_5\}$ have the neighbors v_2 and v_4 in G, then $G \setminus v_3$ contains an induced subgraph isomorphic to a graph in Ω_N.*

Proof of Proposition 4.4. Suppose that $G \setminus T$ contains a thread block of size at least $(k+2)(\mu(k)+2)^2+1$. We can find such a thread block in polynomial time using Theorem 2.2. Let $B := B(x, y) = (B, \sigma, \ell)$ be a thread block of size at least $(k+2)(\mu(k)+2)^2+1$. For convenience, let σ' be the ordering obtained from σ by removing the end vertices x and y.

In the following procedure, we mark some vertices of B in order to find an irrelevant vertex in B. We set $Z := \emptyset$. (1) For each v of T, choose the first $k+2$ vertices z of σ' that are neighbors of v with $R \in \ell(z)$, and choose the last $k+2$ vertices z of σ' that are neighbors of v with $L \in \ell(z)$, and add them to Z. (2) For each pair of two vertices v, v' in T, choose $k+2$ common neighbors of v and v' in B, and add them to Z. (3) Choose the first $k+2$ vertices z of σ' with $R \in \ell(z)$, and choose the last $k+2$ vertices z of σ' with $L \in \ell(z)$, and add them to Z. In each case, If there are at most $k+1$ such vertices, then we add all of them to Z. Then

$$|Z| \leq |T|(2k+4) + |T|^2(k+2) + (2k+4) \leq (k+2)(\mu(k)+2)^2 - 2.$$

Since $|V(B)| \geq (k+2)(\mu(k)+2)^2+1$, there exists a vertex w in $V(B) \setminus Z \setminus \{x, y\}$. We claim that w is an irrelevant vertex. If (G, k) is a Yes-instance, then $(G \setminus w, k)$ is clealry a Yes-instance.

Suppose that $(G \setminus w, k)$ is a Yes-instance and let $X \subseteq V(G) \setminus \{w\}$ such that $|X| \leq k$ and $G \setminus (X \cup \{w\})$ is a thread graph. We may assume that $G \setminus X$ is not a thread graph. So, $G \setminus X$ must have an obstruction in Ω_T that contains the vertex w. Let $X' \subseteq X \cup \{w\}$ be a minimal hitting set for Ω_N in G. From the property of the set T, X' is a minimal hitting set for Ω_N in $G[T]$, which implies that $X' \subseteq T$. Thus $G \setminus X$ must have an induced cycle of length at least 9 that contains w. Let C be an induced cycle of length at least 9 containing w in $G \setminus X$.

We will find an induced subgraph of $G \setminus (X \cup \{w\})$ that is isomorphic to a graph in Ω_N using Lemma 4.5, which leads a contradiction. Let v_1, v_2, w, v_3, v_4 be the consecutive vertices on C. To apply Lemma 4.5, we will find two vertices that are adjacent to v_2 and v_3.

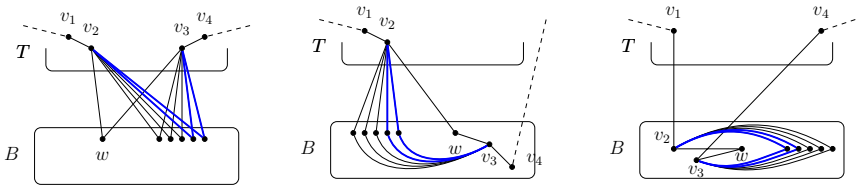


Figure 4 Cases 1–3 in Proposition 4.4.

1. (Case 1. $v_2, v_3 \in T$.) Since v_2 and v_3 have a common neighbor w in $V(B) \setminus Z$, Z contains $k+2$ common neighbors of v_2 and v_3. Since $|X| \leq k$, there exist two vertices $w_1, w_2 \in Z \setminus X$ that are common neighbors of v_2 and v_3.
2. (Case 2. One of v_2 and v_3 is contained in T.) From the symmetry, we may assume that $v_2 \in T$ and $v_3 \notin T$. Since $w \notin \{x, y\}$, v_3 is contained in B. If $R \in \ell(w)$ and $w <_\sigma v_3$, then Z contains the first $k+2$ vertices z of σ' that are neighbors of v_2 with $R \in \ell(z)$. We choose two vertices of them that are not in X. In case when $L \in \ell(w)$ and $v_3 <_\sigma w$, we use the last $k+2$ vertices z of σ' that are neighbors of v_2 with $L \in \ell(z)$ to identify two vertices similarly.
3. (Case 3. Neither v_2 nor v_3 is contained in T.) Since $w \notin \{x, y\}$, v_2 and v_3 are contained in B. If $v_2 <_\sigma w <_\sigma v_3$, then $R \in \ell(v_2)$, $L \in \ell(v_3)$ and it implies that $v_2v_3 \in E$, which is contradiction. Also, $v_3 <_\sigma w <_\sigma v_2$ cannot happen. Thus, both of v_2 and v_3 appear either before w in σ or after w in σ. By the symmetry, we may assume that v_2 and v_3 appear before w in σ. So, $R \in \ell(v_2)$, $R \in \ell(v_3)$, and $L \in \ell(w)$. Since Z contains the last $k+2$ vertices z of σ' with $L \in \ell(z)$, there exist two vertices w_1, w_2 from those $k+2$ vertices that are not in X and C.

In all cases, $G \setminus (X \cup \{w\})$ has an induced subgraph isomorphic to a graph in Ω_N by Lemma 4.5. It contradicts to the assumption that $X \cup \{w\}$ is a LRW1-deletion set of G. Therefore, $G \setminus X$ is a thread graph, and we conclude that (G, k) is a YES-instance. ◀

We show that if a vertex v in T has neighbors on 7 distinct blocks, then we can find a subgraph H isomorphic to one of $\{\beta_1, \beta_2, \beta_3, \beta_4\}$ such that $V(H) \cap T = \{v\}$. However, there is no obstruction in Ω_N containing exactly one vertex from T by Reduction Rule 2. Thus, if a component of $G \setminus T$ has many thread blocks, then we can identify a sequence of consecutive thread blocks not touched by any obstruction in Ω_N. This allows us to contract one of these "safe" thread blocks, say $B(x, y)$, to a vertex v such that $N_{G \setminus T}(v) = (N_{G \setminus T}(x) \cup N_{G \setminus T}(y)) \setminus B(x, y)$.

▶ **Lemma 4.6.** *If $G \setminus T$ has a connected component with at least $19(6\mu(k)+1)$ thread blocks, then we can in polynomial time transform G into a graph $G' = (V', E')$ with $|V'| < |V|$ such that (G, k) is a* YES*-instance if and only if (G', k) is a* YES*-instance.*

4.3 Kernel size

We bound the number of connected components using the following lemma.

▶ **Lemma 4.7.**

1. *The graph $G \setminus T$ has at most $2\mu(k)$ connected components containing at least two vertices.*
2. *If $G \setminus T$ has at least $\mu(k)^2 \cdot (k+2) + 1$ isolated vertices, then we can find an irrelevant vertex in polynomial time.*

Let us now piece everything together and analyze the kernel size.

▶ **Theorem 4.8.** *The* LRW1-Vertex Deletion *problem has a kernel of size* $\mathcal{O}(k^{33})$.

Proof. Let $(G = (V, E), k)$ be an instance of LRW1-Vertex Deletion. By Reduction Rule 1, we may safely assume that G has no components that are thread graphs. Let $T \subset V$ be a vertex subset satisfying Lemma 4.2, and we preprocess using Reduction Rule 2.

By Lemma 4.3, we may assume that for every vertex subset $S \subseteq V$ such that $G[S]$ is a graph of $\{\beta_1, \beta_2, \beta_3, \beta_4\}$, $|S \cap T| \geq 2$. Combining Proposition 4.4 and Lemma 4.6, we can assume that every connected component of $G \setminus T$ has size at most $(k+2)(\mu(k)+2)^2 \cdot 19(6\mu(k)+1)$ (otherwise the instance can be reduced in polynomial time). Note that for each connected component H of $G \setminus T$, there exists a vertex in H that has a neighbor in T, otherwise, H is a component of G that is a thread graph. Therefore, by Lemma 4.7, we can assume that the number of non-trivial components of $G \setminus T$ is at most $2\mu(k)$ and the number of isolated vertices in $G \setminus T$ is at most $\mu(k)^2(k+2)$. It follows that

$$\begin{aligned} |T| + |V \setminus T| &\leq \mu(k) + \big(2\mu(k) \cdot 19(6\mu(k)+1) \cdot (k+2)(\mu(k)+2)^2 + \mu(k)^2 \cdot (k+2)\big) \\ &= \mathcal{O}(k \cdot \mu(k)^4) = \mathcal{O}(k^{33}). \end{aligned}$$

◀

5 Concluding remarks

We consider the problem Linear rankwidth-w Vertex Deletion when $w = 1$. A next step is to investigate the problem for bigger w, or for any fixed w. A closely related problem is Rankwidth-w Vertex Deletion, which asks whether G has a vertex subset of size at most k such that $G \setminus S$ has rankwidth at most w. This problem is fixed-parameter tractable for the following reason. Note that any Yes-instance has rankwidth at most $w + k$. Having bounded rankwith can be characterized by a finite list of forbidden vertex-minors [20]. From [11], having a vertex-minor can be expressed in $\mathsf{C}_2\mathsf{MSO}$, i.e., monadic second order logic without edge set quantification where we can express the parity of $|X|$ for a vertex set X. Fixed-parameter tractability follows as a consequence of Courcelle, Makowsky, Rotics [10].

This result can be turned into a constructive algorithm as [20] provides an explicit upper bound on the size of vertex-minor obstructions for rankwidth k. However, the exponential blow-up in the runtime is huge with respect to both w and k. It is a challenging question whether a reasonable dependency on k can be achieved. A single-exponential time would be ideal, which was achievable for its treewidth counterpart. A first realistic goal is to consider the case when $w = 1$, i.e. the Distance-Hereditary Vertex Deletion. We leave it as an open question whether this problem can be solved in time $c^k \cdot n^{\mathcal{O}(1)}$ time for some constant c.

References

1 Isolde Adler, Arthur M. Farley, and Andrzej Proskurowski. Obstructions for linear rank-width at most 1. *Discrete Applied Mathematics*, 168:3–13, 2014.

2 Isolde Adler, Mamadou Moustapha Kanté, and O-joung Kwon. Linear rank-width of distance-hereditary graphs. In *International Workshop Graph-Theoretic Concepts in Computer Science – WG*, volume 8747 of *Lecture Notes in Computer Science*, pages 42–55, 2014.

3 Hans-Jürgen Bandelt and Henry M. Mulder. Distance-hereditary graphs. *Journal of Combinatorial Theory, Series B*, 41(2):182–208, 1986.

4 Binh-Minh Bui-Xuan, Mamadou Moustapha Kanté, and Vincent Limouzy. A note on graphs of linear rank-width 1. *CoRR*, abs/1306.1345, 2013.

5 Yixin Cao. Unit interval editing is fixed-parameter tractable. In *International Colloquium on Automata, Languages, and Programming – ICALP*, volume 9134 of *Lecture Notes in Computer Science*, pages 306–317, 2015.

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

7 Václav Chvátal and Peter L. Hammer. *Studies in integer programming*, chapter Aggregation of inequalities in integer programming, pages 145–162. Number 1 in Annals of Discrete Mathematics. North-Holland, 1977.

8 Bruno Courcelle. The Monadic Second-Order Theory of Graphs. I. Recognizable Sets of Finite graphs. *Information and Computation*, 85:12–75, 1990.

9 Bruno Courcelle and Mamadou Moustapha Kanté. Graph operations characterizing rank-width. *Discrete Applied Mathematics*, 157(4):627–640, 2009.

10 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.

11 Bruno Courcelle and Sang-il Oum. Vertex-minors, monadic second-order logic, and a conjecture by Seese. *Journal of Combinatorial Theory, Series B*, 97(1):91–126, 2007.

12 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, 2012.

13 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar f-deletion: Approximation, kernelization and optimal FPT algorithms. In *Annual IEEE Symposium on Foundations of Computer Science – FOCS*, pages 470–479, 2012.

14 Fedor V. Fomin, Saket Saurabh, and Yngve Villanger. A polynomial kernel for proper interval vertex deletion. *SIAM Journal on Discrete Mathematics*, 27(4):1964–1976, 2013.

15 Markus Frick and Martin Grohe. The complexity of first-order and monadic second-order logic revisited. *Annals of Pure and Applied Logic*, 130(1-3):3–31, 2004.

16 Robert Ganian. Thread graphs, linear rank-width and their algorithmic applications. In *International Workshop on Combinatorial Algorithms – IWOCA*, volume 6460 of *Lecture Notes in Computer Science*, pages 38–42, Heidelberg, 2011.

17 Robert Ganian and Petr Hlinený. On parse trees and myhill-nerode-type tools for handling graphs of bounded rank-width. *Discrete Applied Mathematics*, 158(7):851–867, 2010.

18 Serge Gaspers and Stefan Szeider. Backdoors to satisfaction. In *The Multivariate Algorithmic Revolution and Beyond – Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*, volume 7370 of *Lecture Notes in Computer Science*, pages 287–317, 2012.

19 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. In *International Colloquium on Automata, Languages, and Programming – ICALP*, volume 7965 of *Lecture Notes in Computer Science*, pages 613–624, 2013.

20 Sang-il Oum. Rank-width and vertex-minors. *Journal of Combinatorial Theory, Series B*, 95(1):79–100, 2005.

21 Sang-il Oum and Paul Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B*, 96(4):514–528, 2006.

22 Geevarghese Philip, Venkatesh Raman, and Yngve Villanger. A quartic kernel for pathwidth-one vertex deletion. In *International Workshop on Graph Theoretic Concepts in Computer Science – WG*, volume 6410 of *Lecture Notes in Computer Science*, pages 196–207, 2010.

23 Neil Robertson and P. D. Seymour. Graph minors. XX. Wagner's conjecture. *J. Combin. Theory Ser. B*, 92(2):325–357, 2004.

24 René van Bevern, Christian Komusiewicz, Hannes Moser, and Rolf Niedermeier. Measuring indifference: unit interval vertex deletion. In *International Workshop on Graph Theoretic Concepts in Computer Science – WG*, volume 6410 of *Lecture Notes in Computer Science*, pages 232–243, 2010.

25 Pim van't Hof and Yngve Villanger. Proper interval vertex deletion. *Algorithmica*, 65(4):845–867, 2013.

Extending the Kernel for Planar Steiner Tree to the Number of Steiner Vertices

Ondřj Suchý*

Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, Czech Republic
ondrej.suchy@fit.cvut.cz

Abstract

In the Steiner Tree problem one is given an undirected graph, a subset T of its vertices, and an integer k and the question is whether there is a connected subgraph of the given graph containing all the vertices of T and at most k other vertices. The vertices in the subset T are called terminals and the other vertices are called Steiner vertices. Recently, Pilipczuk, Pilipczuk, Sankowski, and van Leeuwen [FOCS 2014] gave a polynomial kernel for Steiner Tree in planar graphs, when parameterized by $|T| + k$, the total number of vertices in the constructed subgraph.

In this paper we present several polynomial time applicable reduction rules for Planar Steiner Tree. In an instance reduced with respect to the presented reduction rules, the number of terminals $|T|$ is at most quadratic in the number of other vertices k in the subgraph. Hence, using and improving the result of Pilipczuk et al., we give a polynomial kernel for Steiner Tree in planar graphs for the parameterization by the number k of Steiner vertices in the solution.

1998 ACM Subject Classification E.4 Coding and Information Theory, F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases Steiner Tree, polynomial kernel, planar graphs, polynomial-time preprocessing, network sparsification

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.151

1 Introduction

The Steiner problem is a classical problem of theoretical computer science and a fundamental problem of network design. Most generally it can be formulated as follows: Given a set of interesting objects in some environment or a network, determine the most efficient way to connect them. The study of Steiner Tree in graphs goes back to Hakimi [24] (the problem was also independently formulated by Levin [32]), who showed that Clique can be reduced to Steiner Tree (the theory of NP-hardness was not known yet). Applications can be found in VLSI routing [29], network routing in general [31], phylogenetic tree reconstruction [27] and other areas. It was also the topic of the 11th DIMACS Implementation Challenge. We refer the reader to one of many books devoted to Steiner Tree for further applications [10, 20, 37].

Our focus in this paper is on the case of planar graphs. Hence, we consider the following formulation of the problem:

Planar Steiner Tree
Input: A planar graph $G = (V, E)$, a set $T \subseteq V$, and an integer k.
Question: Is there a set $S \subseteq V \setminus T$ of size $|S| \leq k$ such that $G[T \cup S]$ is connected?

* The research was partially supported by the grant 14-13017P of the Czech Science Foundation.

10th International Symposium on Parameterized and Exact Computation (IPEC 2015).
Editors: Thore Husfeldt and Iyad Kanj; pp. 151–162

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

We denote n the number of vertices of the graph G and m the number of its edges. We call the vertices in T *terminals* and the vertices in $V \setminus T$ *non-terminals* or *Steiner vertices*. We call a set $S \subseteq V \setminus T$ a solution (for the instance (G, T, k)), if $|S| \leq k$ and for every pair of vertices $x, y \in T$ the vertices x and y are in the same connected component of $G[S \cup T]$. Note that then all vertices of T are in the same connected component of $G[S \cup T]$ and we can remove all other connected components from the graph and get a set asked by the problem definition.

The problem is often formulated so that one is to find a connected subgraph with the minimum number of edges that contains the set of terminals. The subgraph attaining the minimum must be a tree, called *optimal Steiner tree*. It is not hard to see that there is such a connected subgraph with $|T| + k - 1$ edges if and only if there is a set of vertices forming a solution to our formulation (with k Steiner points).

Our Contribution

The problem is NP-hard [23] and remains so even in very restricted planar cases [22]. In order to better understand the complexity of the problem we focus on the parameterized analysis of the problem. The problem was recently studied with respect to the parameter "the number of edges in an optimal Steiner tree" or equivalently $|T| + k$ by Pilipczuk, Pilipczuk, Sankowski, and van Leeuwen [35, 36], who obtained a subexponential algorithm [35] and a polynomial kernel [36] for the problem with respect to this parameterization. In particular they proved the following proposition.

▶ **Proposition 1** (Pilipczuk et al. [36]). *Given a* Planar Steiner Tree *instance* (G, T)*, one can in* $O(k_{OPT}^{142} n)$ *time find a set* $F \subseteq E(G)$ *of* $O(k_{OPT}^{142})$ *edges that contains an optimal Steiner tree connecting* T *in* G*, where* k_{OPT} *is the total number of edges of an optimal Steiner tree.*

In this paper we focus on the parameterization by the number k of Steiner vertices in the solution. By folklore result Planar Steiner Tree is known to be fixed parameter tractable with respect to this parameter (see also [28]), however it was not known whether there is a polynomial kernel. We resolve this question as follows: We present several polynomial time reduction rules and show that if the rules are exhaustively applied the number of terminals is at most quadratic in k. Then the number of edges of an optimal Steiner tree of that instance is also at most quadratic in k and we can use the algorithm of Proposition 1 to obtain a polynomial kernel.

This improves the result of Pilipczuk et al. qualitatively, since we give a polynomial kernel with respect to a parameter that is always smaller and can be arbitrarily smaller than the parameter they use. Moreover, it also improves it quantitatively, as our rules never increase the number of edges in an optimal Steiner tree, and, hence, the kernel obtained by first running our rules is always at most as big as the one obtained by starting directly with the algorithm of Proposition 1.

Related Work

As we already mentioned the problem is NP-complete even in very restricted cases of planar graphs [22]. It is also well studied from the approximation perspective. It can be approximated to within a factor $O(\log n)$, but it cannot be approximated within a factor $(1 - \varepsilon)(\log |T|)$ unless NP $\subseteq$ DTIME$[n^{\text{polylog}(n)}]$ [30]. Furthermore, the edge weighted variant admits a constant factor approximation [9], while it is APX-complete even on complete graphs with weights 1 and 2 [2].

Many studies considered the planar variant of the problem from the approximation perspective. The edge weighted variant admits EPTAS on planar graphs [8] as well as on bounded genus graphs [7]. The number of papers on approximation of some variant of Steiner type problem is enormous and rapidly growing. Therefore we refer the reader to the online compendium [25] for the current approximation state of various Steiner type problems.

Turning to exact exponential algorithms, there is a simple folklore algorithm running in $O(2^{\frac{2}{3}n}n^{O(1)}) = O(1.6181^n)$ time. This was improved to $O(1.59^n)$ for the weighted case and $O(1.36^n)$ for the cardinality case by Fomin et al. [17]. Note that all of the mentioned algorithms use polynomial space.

Concerning parameterized complexity, on general graphs, Steiner Tree is known to be W[2]-hard with respect to the standard parameterization k [14]. Moreover, by result of Patrascu and Williams [34], there are no $l \geq 3$, $\varepsilon > 0$, and an algorithm that would solve Steiner Tree on instance (G, T, k) with $k = l$ in time $O(n^{l-\varepsilon})$, unless the Strong Exponential Time Hypothesis (SETH) fails.

However, for Steiner Tree much more often the parameterization by the number of terminals $|T|$ is used. There is a nice long history of improving FPT-algorithms with respect to this parameterization started by the $O(3^{|T|} \cdot n + 2^{|T|} \cdot n^2 + n(n \log n + m))$-time algorithm of Dreyfus and Wagner [15] (independently found by Levin [32]). This algorithm, as well as its later improvements [16, 21, 4] subsequently approaching the $O^*(2^{|T|})$ running time, use exponential space.

Polynomial space FPT-algorithms appeared only recently. The one by Nederlof [33] applies to the cardinality variant and the variant where the weights are bounded by a constant and achieves $O^*(2^{|T|})$ time. The running time of $O^*(2^{|T|})$ is believed to be optimal [11]. The algorithm by Fomin et al. [18] achieves running time $O(7.97^{|T|} \cdot n^4 \cdot \log W)$ for edge weights in $\{1, \ldots, W\}$.

On general graphs the problem does not admit polynomial kernel even with respect to $k + |T|$, unless coNP $\subseteq$ NP/poly. This can be easily proved using the framework of Bodlaender et al. [6], a less direct approach can be found in [13]. Note also that since the problem is on general graphs FPT with respect to $|T|$ and W[2]-hard with respect to k, no reduction of type presented in our paper can exist for general graphs, unless W[2]=FPT.

By way of contrast, much less is known about the parameterized complexity of Planar Steiner Tree. In fact, for the parameterization by $|T|$, no results are known that would improve those from general graphs. On the other hand, for the parameterization by k, the problem is known to be FPT by a folklore result based on Steiner Tree being FPT with respect to the treewidth of the graph G [12, 5, 19]. This was improved by Jones et al. [28], who presented an algorithm in $O^*(3^{dk+o(dk)})$ that applies to d-degenerate graphs.

However, recently there was a significant progress for planar graphs with respect to the combined parameter $k + |T|$. First Pilipczuk et al. [35] showed that there is a subexponential algorithm for Planar Steiner Tree with respect to this parameterization running in $O(2^{O(((k+|T|)\log(k+|T|))^{2/3})}n)$ time. Later the same group of authors improved this to $O(2^{O(\sqrt{(k+|T|)\log(k+|T|)})}n)$ time and gave the kernel as presented in Proposition 1 [36].

Organization of the paper

Our algorithm is described in Section 2. In particular Subsection 2.1 contains the reduction rules that apply to all graphs, Subsection 2.2 collects the rules that only hold in planar graphs and Subsection 2.3 contains the main theorem and the analysis of the running time. We conclude the paper by giving some outlooks for future research in Section 3.

Due to space constraints, several proofs had to be deferred to the full version of the paper.

2 Algorithm

Our algorithm consist of several reduction rules, which simplify the instance. The algorithm applies these rules exhaustively to obtain an instance which is *reduced* with respect to them, that is, an instance to which none of the rules applies any more. We give the rules in a specific order and always prefer to apply a rule that was given earlier to a rule given later. In other words, before we apply some rule, we assume that the instance is reduced with respect to all previous rules.

For each of the rules we immediately prove its *correctness*, that is, the instance produced by the rule is a yes-instance if and only if the original instance was (the instances are *equivalent*). However we defer the analysis of the running times to find applications and to apply the rules until all the rules are presented.

We present the rules and some auxiliary lemmata that apply to general graphs in Subsection 2.1, rules for planar graphs in Subsection 2.2 and the running time and the main theorem in Subsection 2.3.

2.1 General Reduction Rules

Here we give the rules and lemmata that would apply to general graphs, however also preserve planarity of the instance. We start by a rule that summarizes the obvious trivial constraints on solvability of the instance. Its correctness is immediate.

▶ **Reduction Rule 1.**
(a) If $k \geq 0$ and $\emptyset$ is a solution, then answer YES.
(b) If $k < 0$, then answer NO.
(c) If $k = 0$ and $\emptyset$ is not a solution, then answer NO.
(d) If for some $x, y \in T$ there is no path between x and y in G, then answer NO.

We continue by a well known rule for Steiner Tree.

▶ **Reduction Rule 2** (Folklore)**.** If there are two adjacent terminals x and y, then contract the edge $\{x, y\}$. I.e., we continue with the instance (G', T', k), where $G' = (V', E')$, $V' = (V \setminus \{x, y\}) \cup \{w\}; w \notin V$, $E' = (E \setminus \{e \mid e \in E, e \cap \{x, y\} \neq \emptyset\}) \cup \{(e \setminus \{x, y\}) \cup \{w\} \mid e \in E, |e \cap \{x, y\}| = 1\})$, and $T' = (T \setminus \{x, y\}) \cup \{w\}$.

Although the rule is well known, we prove its correctness for completeness.

▶ **Lemma 2** (★[1])**.** *Reduction Rule 2 is correct.*

The following lemma pinpoints the property of solutions that is crucial for our considerations in the rest of the paper. We use the following notion: a Steiner vertex v *dominates* a terminal x if v and x are adjacent in G.

▶ **Lemma 3.** *Let (G, T, k) be an instance where Reduction Rules 1 and 2 have been exhaustively applied, S a solution of the instance, and x a vertex in T. Then there is a vertex v in S such that v dominates x.*

Proof. Since the instance is reduced with respect to Reduction Rule 1, we know that there are at least two terminals, as otherwise $\emptyset$ would be a solution. Let y be a terminal different from x. Since S is a solution, there is a path $p_1, p_2, \ldots, p_q$ in $G[S \cup T]$ such that $p_1 = x$ and

[1] Proofs of lemmata marked with (★) were deferred to the full version of the paper.

$p_q = y$. Let $v = p_2$. If v is in T, then x and v are two adjacent terminals contradicting the instance being reduced with respect to Reduction Rule 2. Hence v is in S and dominates x as claimed. ◀

We will use the following lemma in our proofs to show that taking some vertex to a solution can be modeled by a suitable modification of the instance and this preserves yes-instances.

▶ **Lemma 4.** *Let (G, T, k) be an instance, S a solution for (G, T, k), and v a vertex in S such that there is a terminal in the same connected component of $G[T \cup S]$ as v. Let us denote $T' = T \cup \{v\}$ and $k' = k - 1$. Then (G, T', k') is a yes-instance.*

The condition on v being in the same connected component of $G[T \cup S]$ as some terminal might seem a little strange and it would be more natural to just assume S to be minimal. However, we prefer to formulate the lemma this way, as it makes it easier to use.

Proof. Let us denote $S' = S \setminus \{v\}$. Then $|S'| \leq k'$ and $T \cup S$ equals $T' \cup S'$. Thus for every pair of vertices y, z from T the terminals y and z are in the same connected component of $G[T' \cup S'] = G[T \cup S]$. Moreover, v is in the same connected component of $G[T' \cup S'] = G[T \cup S]$ as at least one other terminal and, thus, all other terminals, by assumption. Hence S' is a solution for (G, T', k'), finishing the proof. ◀

The following lemma is complementary to the previous one and shows that the operation preserves no-instances.

▶ **Lemma 5.** *Suppose (G, T, k) is an instance, $v \in V \setminus T$, and $x \in T$. Let us denote $T' = T \cup \{v\}$ and $k' = k - 1$. If (G, T', k') is a yes-instance, then (G, T, k) is a yes-instance.*

Proof. Let S' be a solution for (G, T', k') and let us denote $S = S' \cup \{v\}$. Then $|S| \leq k$ and again $T \cup S$ equals $T' \cup S'$ and, thus, $G[T \cup S] = G[T' \cup S']$. Since $T \subseteq T'$, S is a solution for (G, T, k), finishing the proof. ◀

The following rule shows that (false) twins among the terminals are superfluous. Surprisingly, we were not able to find the use of such a rule for Steiner Tree in literature.

▶ **Reduction Rule 3.** If there are x and y in T such that $N(x) = N(y)$, then remove x from G. I.e., we continue with instance (G', T', k), where $G' = G \setminus x$ and $T' = T \setminus \{x\}$.

▶ **Lemma 6 (★).** *Reduction Rule 3 is correct.*

2.2 Plane Specific Reduction Rules

In this subsection we present rules that rely on the graph being planar. To ease the presentation it is better to fix an embedding of the graph. Hence, for the rest of the paper we assume, that the given graph G is plane, i.e., it has a fixed planar embedding. Since a planar embedding can be found in linear time [26], this assumption is not restrictive. In fact we often consider the sphere embedding, as we mostly do not distinguish the outer face.

Consider two non-terminals u and v with at least two common terminal neighbors. Denote the set of common terminal neighbors Q and $q = |Q|$. The embedding of the edges connecting the vertices u and v to the vertices of Q cuts the surface of the sphere into q connected areas. Let A be any of these areas. If x and y are the two vertices of Q incident to the area A, then we say that u, x, v, and y form an *eye* and the cycle u, v, x, y forms its *boundary*. Moreover, we say that a vertex is *inside* the eye u, x, v, y, if it is embedded inside the area A. We say that a vertex is *outside* the eye if it is neither inside nor on the boundary of it. Note that a

terminal z inside the eye u, x, v, y can be connected to u or v, but not to both of them by the definition of an eye.

Our first planar rule applies to eyes where one of the Steiner points on the boundary dominates all the terminals inside the eye.

▶ **Reduction Rule 4.** Suppose u, x, v, and y form an eye, such that $u, v \in V \setminus T$, and $x, y \in T$. Further suppose that there is a terminal $z \in T$ inside the eye. If every terminal inside the eye is a neighbor of v, then add v to T and reduce k by one.

The intuition behind the proof of correctness is that taking v into the solution is always at least as good as taking any vertex inside the eye.

▶ **Lemma 7 (★).** *Reduction Rule 4 is correct.*

The next rule allows to make an eye which does not contain terminals completely empty. We require the eye to be non-empty before the application of the rule so that the application actually changes the graph.

▶ **Reduction Rule 5.** Suppose u, x, v, and y form an eye, such that $u, v \in V \setminus T$ and $x, y \in T$ and suppose that there is a vertex $w \in V \setminus T$, but no terminal, inside the eye. Then remove every vertex w inside the eye from G.

Intuitively, if any vertex inside the eye is to be in the solution, then we can take u (or v) and detour any path around the eye.

▶ **Lemma 8 (★).** *Reduction Rule 5 is correct.*

The following variant of the so-called "high degree rule" forms the crux of our algorithm.

▶ **Reduction Rule 6.** If there is a vertex $u \in V \setminus T$ which dominates more than $5k$ terminals, then add u to T and reduce k by one. I.e., continue with instance (G, T', k'), where $k' = k - 1$ and $T' = T \cup \{u\}$.

▶ **Lemma 9.** *Reduction Rule 6 is correct.*

Proof. By Lemma 5, if the resulting instance is a yes-instance, then so is the original one. Now for the other direction, assume that there is a set $S \subseteq V \setminus T$ that is a solution for (G, T, k). If S contains u, then the resulting instance is a yes-instance by Lemma 4. Hence, let us assume for the rest of the proof that $u \notin S$. We show that this leads to a contradiction with the instance being reduced with respect to the previous rules.

Consider the terminals in $N(u)$. Each of them is dominated by a vertex of S by Lemma 3. Let B be the set of vertices in S dominating at least 2 vertices in $N(u) \cap T$. Each vertex in B forms at least 2 eyes together with u. We want to show that there must be a vertex in B for which many of these eyes are empty. To this end, let us fix a face of the embedding of G as the outer one and call an eye *outer* if it contains the outer face and *internal* otherwise. We will use the following auxiliary claim about the eyes between u and vertices in B.

▶ **Claim 1 (★).** *There are no two internal eyes u, x, b, y and u, x', c, y' with $b \neq c$ such that the union of their boundaries cuts the plane into 4 regions.*

Let us define the following relation on vertices of S. For $a, b \in S$ we write $a < b$ if and only if b is in B and a is inside some internal eye formed by u and b. We show that this relation can be used to order the vertices of S.

▶ **Claim 2 (★).** *The relation $<$ is a strict partial order.*

Let us now define another relation $\prec$ as follows. We write $a \prec b$ if and only if $a < b$ and there is no c such that $a < c$ and $c < b$. This relation is the cover relation of the strict partial order $<$. We need the following property of that relation.

▶ **Claim 3 (★).** *For every a there is at most one b such that $a \prec b$.*

For a vertex b in B we call the set of vertices a such that $a \prec b$ the *support* of b and denote $\mathrm{supp}(b)$. The support gives each vertex of B a budget for nonempty eyes in the following sense.

▶ **Claim 4 (★).** *Let b be a vertex in B and consider an eye A between u and b containing a vertex of S. Then there is a vertex a inside A such that $a \prec b$ and, hence, a is in $\mathrm{supp}(b)$.*

Next we show that, since the budget is limited, there are vertices which dominate more vertices than what their budget allows them.

▶ **Claim 5.** *There is a vertex b of B which dominates more than $2|\mathrm{supp}(b)| + 3$ vertices of $T \cap N(u)$.*

Proof (of Claim 5). Suppose for contradiction that each vertex b in B dominates at most $2|\mathrm{supp}(b)| + 3$ vertices of $T \cap N(u)$. Then, since every vertex a in $S \setminus B$ dominates at most 1 vertex of $T \cap N(u)$ by definition, we have

$$\begin{aligned} 5k < |T \cap N(u)| &\leq \sum_{b \in B} (2|\mathrm{supp}(b)| + 3) + \sum_{a \in S \setminus B} 1 \leq 3k + 2 \sum_{b \in B} |\{a \mid a \prec b\}| \\ &= 3k + 2 \sum_{a \in S} |\{b \mid a \prec b\}| \leq 3k + 2k = 5k \end{aligned}$$

which is a contradiction. ◀

Let v be a vertex which dominates more than $2|\mathrm{supp}(v)| + 3$ vertices of $T \cap N(u)$. Let us denote C the set of terminals that are common neighbors of u and v. We know, that there are $|C|$ eyes between u and v. For each internal eye which contains a vertex of S there is vertex in $\mathrm{supp}(v)$ by Claim 4. Therefore, there are at most $|\mathrm{supp}(v)| + 1$ eyes which contain a vertex of S inside. We show, that the eyes that do not contain any vertex of S do not contain any vertices at all.

Assume to the contrary that there is an eye u, x, v, y that does not contain any vertex of S, but its interior is not empty. If there was no terminal in the interior, then Reduction Rule 5 would apply, contradicting the instance being reduced with respect to the previous rules. Hence, the set of terminals inside the eye is non-empty. Let us denote it T_e. By Lemma 3, each terminal has to be dominated by a vertex of S. Since there are no vertices of S inside the eye and $u \notin S$ by assumption, it follows that $T_e \subseteq N(v)$ and Reduction Rule 4 would apply, again contradicting the instance being reduced with respect to the previous rules.

Hence, each eye not containing any vertex of S is empty. Therefore there are at most $|\mathrm{supp}(v)| + 1$ nonempty eyes which have together at most $2|\mathrm{supp}(v)| + 2$ terminals on their boundaries. Since C contains more than $2|\mathrm{supp}(v)| + 3$ vertices it follows that there are two vertices x and y in C that only appear on boundaries of empty eyes and, hence, are of degree two. In particular, for their neighborhoods we have $N(x) = N(y) = \{u, v\}$ and Reduction Rule 3 would apply, contradicting the instance being reduced with respect to the previous rules. Hence, if the original instance was a yes-instance, then so is the resulting one, finishing the proof of the lemma. ◀

Equipped with Reduction Rule 6 it is easy to finally bound the number of terminals in the instance.

▶ **Reduction Rule 7.** If there is more than $5k^2$ terminals, then answer NO.

▶ **Lemma 10.** *Reduction Rule 7 is correct.*

Proof. Suppose for contradiction that $|T| > 5k^2$, but the instance is a yes instance. Let S be a solution. By Lemma 3 each vertex of T has at least one neighbor in S. On the other hand, $|S| \leq k$ and, hence, by pigeonhole principle there is a vertex in S with more than $5k$ neighbors in T. But this contradicts the instance being reduced with respect to Reduction Rule 6. ◀

2.3 Main Theorem and Time Complexity

In this subsection we piece together our main result. We start by analyzing the time needed to exhaustively apply the reduction rules.

▶ **Lemma 11.** *Given an instance (G, T, k) of* Planar Steiner Tree *one can in $O(n^4)$-time either correctly decide it or obtain an equivalent instance (G'', T', k') which is reduced with respect to Reduction Rules 1–7, $k' \leq k$, and $|T'| + k'$ is at most $|T| + k$.*

Proof. We apply the reduction rules exhaustively. The equivalence of resulting instance follows from the correctness of the reduction rules. Here we argue about the running time to apply the reduction rules.

Let us first consider the time spend to find one application and to apply the rule one time for each of the rules.

To check whether Reduction Rule 1 can be applied one can in linear time find the connected components of graph G and graph $G[T]$ and then check whether all the terminals are in the same connected components. Therefore, to check whether the rule applies we need $O(n)$ time. The rule can be applied in constant time, as we directly answer.

We can find an application of Reduction Rule 2 by going through the edges of the graph in linear time. The modification of the instance also takes linear time.

To check whether Reduction Rule 3 applies we can determine for every pair of vertices x, y whether their neighborhoods are equal in cubic time. The instance can again be modified in linear time.

To apply Reduction Rule 4 and Reduction Rule 5 we first need to find the embedding, the eyes, and which vertices are inside them. We try all pairs of non-terminals u, v and for each of them we find the eyes formed between them (if any) in linear time. Then in linear time we find for all other vertices inside which eye they are and finally whether the rules apply to any of the eyes. Summing over all pairs u, v, an application of the rule can be found in $O(n^3)$ time. The application takes constant time for Reduction Rule 4 and linear time for Reduction Rule 5.

An application of Reduction Rule 6 or Reduction Rule 7 can be found and the rule applied in linear time.

Now we would like to bound the number of times we search for an application of a rule and the number of times a rule is applied. Note that after any rule is applied as well as on the beginning we iterate through the reduction rules starting from the first one and continuing with the later ones and we stop when we find no application of a rule anymore. Hence the number of times we search for an application of a particular rule is linear in the number of applications of the rules.

If Reduction Rule 1 or 7 is applied, then the algorithms stops. Hence, these rules are only applied once. Reduction Rules 2, 3, and 5 reduce the number of vertices of the graph. Since no rule increases the number of vertices, there can be at most n application of these rules together. Finally, Reduction Rules 4 and 6 reduce the number of non-terminals (by converting them to terminals) and, hence, there can be at most n applications of these two rules together.

Summing up, the rules can be exhaustively applied in $O(n^4)$ time and the lemma follows by observing that no rule increases k or $k + |T|$. ◀

▶ Remark. It seems possible to improve the running time given in Lemma 11, e.g., Reduction Rule 3 only has to be applied for vertices of degree 2, etc. However, this would make analysis more complicated and we prefer to focus this extended abstract in different direction.

Now we are ready to prove our main theorem, which combines Lemma 11 with Proposition 1, to obtain the kernel for Planar Steiner Tree with respect to k.

▶ **Theorem 12.** *Given a* Planar Steiner Tree *instance* (G, T, k) *one can in* $O(k^{284}n+n^4)$ *time either correctly decide decide it or find an equivalent instance* (G', T', k') *with* $k' \leq k$, $|T'| \leq k'^2$, *and* $|V(G')| \leq O(k^{284})$.

Proof. By Lemma 11 we can in $O(n^4)$ time find an equivalent instance (G'', T', k') which is reduced with respect to Reduction Rules 1–7 and $k' \leq k$. Since the instance (G'', T', k') is reduced with respect to Reduction Rule 7, we have $|T'| \leq 5k^2$ and if (G'', T', k') is a yes-instance, then there is an optimal Steiner tree with at most $k' + |T'| = O(k'^2)$ edges.

Now we run the algorithm of Proposition 1 on the instance (G'', T') for $O(k'^{284}n)$ time. If it fails to finish, then the number of edges in an optimal Steiner tree k_{OPT} is more than $k' + |T'| = O(k'^2)$ and we answer NO. If it finishes, then let F be the set of edges returned. If $|F|$ is not $O(k'^{284})$, then we again answer NO. Otherwise, we let G' be the graph (W, F), where W is the set of vertices of G'' which are incident to at least one edge of F. By Proposition 1 and since no terminal is isolated in any Steiner tree, the instances (G'', T', k') and (G', T', k') are equivalent. Since $|F| = O(k_{\text{OPT}}^{142})$ and $k_{\text{OPT}} \leq 5k'^2 + k' = O(k'^2)$, the bound on the size of $V(G')$ follows. ◀

3 Conclusion and Future Directions

We presented the first polynomial kernel for Planar Steiner Tree with respect to the number k of Steiner points in the solution. It seems plausible that the kernel size bound as well as the running time of the algorithm can be improved. First of all, already the size bound $O((k+|T|)^{142})$ given by Proposition 1 probably offers plenty of space for improvements, with the best known lower bound (for the method) being just $\Omega((k + |T|)^2)$.

Second, our approach mostly relies on domination type arguments. Dominating Set in Planar Graphs is known to have a linear kernel [1] and even one such computable in linear time [3]. It might be possible to use the ideas of these kernels to reduce the number of terminals to linear in k or at least reduce the running time to linear. If one was able to reduce the number of terminals to linear in k, this would immediately lead to a subexponential FPT-algorithm for Planar Steiner Tree parameterized by k, which is itself an interesting open problem.

Given the amount of research conducted on Steiner Tree in general graphs with respect to the number of terminals $|T|$, it seems surprising that we cannot say anything more with respect to this parameterization on planar graphs. In particular, we are aware neither of polynomial kernel nor of a subexponential FPT-algorithm for Planar Steiner Tree with respect to this parameterization.

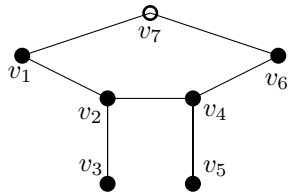


Figure 1 Illustration of different notions of optimality for Steiner forests. For $\mathcal{T} = \{\{v_1, v_6\}, \{v_2, v_3\}, \{v_4, v_5\}\}$, the set of terminals already induces a connected subgraph of G. Hence no Steiner point is needed. On the other hand, the spanning tree of this induced subgraph has 5 edges (and we cannot omit any of them), but it is possible to connect the terminals as required by taking the following 4 edges: $\{v_1, v_7\}, \{v_7, v_6\}, \{v_2, v_3\}, \{v_4, v_5\}$.

Finally, one might want to try to generalize the ideas of this paper to further problems and Planar Steiner Forest seems a natural candidate. Here we are given a graph G and a family of pairs of vertices $\mathcal{T}$ and the task is to find the smallest subgraph of G in which the vertices of each pair from $\mathcal{T}$ appear in the same connected component. While Pilipczuk et al. [36] proved an analogue of Proposition 1 for Planar Steiner Forest (with the degree of the polynomial in the bound 4 times larger), there are several issues preventing such a generalization.

First, for Steiner Forest it is no longer true that the optimal forest is the one obtained as a spanning forest of the subgraph with minimum number of Steiner points (usually all members of all pairs in $\mathcal{T}$ are called terminals), as Figure 1 illustrates. Since the result of Pilipczuk et al. minimizes the number of edges, whereas our formulation optimizes the number of Steiner points, it might be troublesome to combine them.

Second, for Steiner Forest there is no direct analogue of Reduction Rule 2. Indeed, notice that applying Reduction Rule 2 to, e.g., the edge $\{v_2, v_3\}$ of the graph on Figure 1 causes the optimum number of Steiner points in the solution to increase to one, as the vertex resulting from the contraction is no longer a terminal. Moreover the rule also applies to edge $\{v_1, v_2\}$ which is not in the Steiner forest minimizing the number of edges.

We still believe that one might be able to reduce the instance in such a way that the optimum solution only uses $O(k^2)$ edges in trees that contain at least two edges. It is not obvious whether such a reduction could be combined with the ideas of Pilipczuk et al. to obtain a polynomial kernel for Planar Steiner Forest with respect to the number of Steiner points in the solution.

There are also reasons not to believe that such a kernel should exist for Planar Steiner Forest, since the problem behaves significantly different than Planar Steiner Tree. Indeed, Pilipczuk et al. [36] also proved that, in contrast to Planar Steiner Tree, there is no subexponential FPT-algorithm for Planar Steiner Forest parameterized by the total number of edges in an optimal Steiner forest.

Hence, we believe that there are many interesting directions to study related to the current paper.

References

1 J. Alber, M. R. Fellows, and R. Niedermeier. Polynomial-time data reduction for dominating set. *J. ACM*, 51(3):363–384, May 2004.

2 M. W. Bern and P. E. Plassmann. The Steiner problem with edge lengths 1 and 2. *Inf. Process. Lett.*, 32(4):171–176, 1989.

3 R. van Bevern, S. Hartung, F. Kammer, R. Niedermeier, and M. Weller. Linear-time computation of a linear problem kernel for dominating set on planar graphs. In *Parameterized and Exact Computation - 6th International Symposium, IPEC 2011, Revised Selected Papers*, volume 7112 of *LNCS*, pages 194–206. Springer, 2011.

4 A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Fourier meets Möbius: fast subset convolution. In *Proceedings of the 39th ACM Symposium on Theory of Computing, STOC 2007*, pages 67–74. ACM, 2007.

5 H. L. Bodlaender, M. Cygan, S. Kratsch, and J. Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Inf. Comput.*, 243:86–111, 2015.

6 H. L. Bodlaender, R. G. Downey, M. R. Fellows, and D. Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009.

7 G. Borradaile, E. D. Demaine, and S. Tazari. Polynomial-time approximation schemes for subset-connectivity problems in bounded-genus graphs. *Algorithmica*, 68(2):287–311, 2014.

8 G. Borradaile, P. N. Klein, and C. Mathieu. An $O(n \log n)$ approximation scheme for Steiner tree in planar graphs. *ACM Transactions on Algorithms*, 5(3), 2009.

9 J. Byrka, F. Grandoni, T. Rothvoß, and L. Sanità. Steiner tree approximation via iterative randomized rounding. *J. ACM*, 60(1):6:1–6:33, Feb. 2013.

10 D. Cieslik. *Steiner minimal trees*, volume 23. Springer Science & Business Media, 1998.

11 M. Cygan, H. Dell, D. Lokshtanov, D. Marx, J. Nederlof, Y. Okamoto, R. Paturi, S. Saurabh, and M. Wahlström. On problems as hard as CNF-SAT. In *Proceedings of the 27th Conference on Computational Complexity, CCC 2012*, pages 74–84. IEEE, 2012.

12 M. Cygan, J. Nederlof, M. Pilipczuk, M. Pilipczuk, J. M. M. van Rooij, and J. O. Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011*, pages 150–159. IEEE Computer Society, 2011.

13 M. Dom, D. Lokshtanov, and S. Saurabh. Kernelization lower bounds through colors and IDs. *ACM Transactions on Algorithms*, 11(2):13:1–13:20, 2014.

14 R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, New York, 1999.

15 S. E. Dreyfus and R. A. Wagner. The Steiner problem in graphs. *Networks*, 1:195–207, 1972.

16 R. E. Erickson, C. L. Monma, and A. F. Veinott, Jr. Send-and-split method for minimum-concave-cost network flows. *Mathematics of Operations Research*, 12(4):634–664, 1987.

17 F. V. Fomin, F. Grandoni, D. Kratsch, D. Lokshtanov, and S. Saurabh. Computing optimal Steiner trees in polynomial space. *Algorithmica*, 65(3):584–604, 2013.

18 F. V. Fomin, P. Kaski, D. Lokshtanov, F. Panolan, and S. Saurabh. Parameterized single-exponential time polynomial space algorithm for Steiner tree. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Proceedings, Part I*, volume 9134 of *LNCS*, pages 494–505. Springer, 2015.

19 F. V. Fomin, D. Lokshtanov, and S. Saurabh. Efficient computation of representative sets with applications in parameterized and exact algorithms. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014*, pages 142–151. SIAM, 2014.

20 D. S. R. Frank K. Hwang and P. Winter. *The Steiner Tree Problem*, volume 53 of *Annals of Discrete Mathematics*. Elsevier, 1992.

21 B. Fuchs, W. Kern, D. Mölle, S. Richter, P. Rossmanith, and X. Wang. Dynamic programming for minimum Steiner trees. *Theory of Computing Systems*, 41(3):493–500, 2007.

22 M. R. Garey and D. S. Johnson. The rectilinear Steiner tree problem in NP complete. *SIAM Journal of Applied Mathematics*, 32:826–834, 1977.

23 M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.

24 S. L. Hakimi. Steiner's problem in graphs and its implications. *Networks*, 1:113–133, 1971.

25 M. Hauptmann and M. Karpinski. A compendium on Steiner tree problems. online. `http://theory.cs.uni-bonn.de/info5/steinerkompendium/`.

26 J. Hopcroft and R. Tarjan. Efficient planarity testing. *J. ACM*, 21(4):549–568, Oct. 1974.

27 F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner Tree Problem*. North-Holland, Amsterdam, 1992.

28 M. Jones, D. Lokshtanov, M. S. Ramanujan, S. Saurabh, and O. Suchý. Parameterized complexity of directed Steiner tree on sparse graphs. In *Algorithms - ESA 2013 - 21st Annual European Symposium, Proceedings*, volume 8125 of *LNCS*, pages 671–682. Springer, 2013.

29 A. B. Kahng and G. Robins. *On Optimal Interconnections for VLSI*. Kluwer Academic Publisher, 1995.

30 P. Klein and R. Ravi. A nearly best-possible approximation algorithm for node-weighted Steiner trees. *Journal of Algorithms*, 19(1):104 – 115, 1995.

31 B. Korte, H. J. Prömel, and A. Steger. Steiner trees in VLSI-layout. In *Paths, Flows and VLSI-Layout*, pages 185–214, 1990.

32 A. Y. Levin. Algorithm for the shortest connection of a group of graph vertices. *Sov. Math. Dokl.*, 12:1477–1481, 1971.

33 J. Nederlof. Fast polynomial-space algorithms using inclusion-exclusion. *Algorithmica*, 65(4):868–884, 2013.

34 M. Patrascu and R. Williams. On the possibility of faster SAT algorithms. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010*, pages 1065–1075. SIAM, 2010.

35 M. Pilipczuk, M. Pilipczuk, P. Sankowski, and E. J. van Leeuwen. Subexponential-time parameterized algorithm for Steiner tree on planar graphs. In *30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013*, volume 20 of *LIPIcs*, pages 353–364. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

36 M. Pilipczuk, M. Pilipczuk, P. Sankowski, and E. J. van Leeuwen. Network sparsification for Steiner problems on planar and bounded-genus graphs. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014*, pages 276–285. IEEE Computer Society, 2014.

37 H. J. Prömel and A. Steger. *The Steiner Tree Problem; a Tour through Graphs, Algorithms, and Complexity*. Vieweg, 2002.

Sparsification Upper and Lower Bounds for Graphs Problems and Not-All-Equal SAT*

Bart M. P. Jansen and Astrid Pieterse

Eindhoven University of Technology
P. O. Box 513, Eindhoven, The Netherlands
{b.m.p.jansen,a.pieterse}@tue.nl

Abstract

We present several sparsification lower and upper bounds for classic problems in graph theory and logic. For the problems 4-Coloring, (Directed) Hamiltonian Cycle, and (Connected) Dominating Set, we prove that there is no polynomial-time algorithm that reduces any n-vertex input to an equivalent instance, of an arbitrary problem, with bitsize $\mathcal{O}(n^{2-\varepsilon})$ for $\varepsilon > 0$, unless NP $\subseteq$ coNP/poly and the polynomial-time hierarchy collapses. These results imply that existing linear-vertex kernels for k-Nonblocker and k-Max Leaf Spanning Tree (the parametric duals of (Connected) Dominating Set) cannot be improved to have $\mathcal{O}(k^{2-\varepsilon})$ edges, unless NP $\subseteq$ coNP/poly. We also present a positive result and exhibit a non-trivial sparsification algorithm for d-Not-All-Equal-SAT. We give an algorithm that reduces an n-variable input with clauses of size at most d to an equivalent input with $\mathcal{O}(n^{d-1})$ clauses, for any fixed d. Our algorithm is based on a linear-algebraic proof of Lovász that bounds the number of hyperedges in critically 3-chromatic d-uniform n-vertex hypergraphs by $\binom{n}{d-1}$. We show that our kernel is tight under the assumption that NP $\not\subseteq$ coNP/poly.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases sparsification, graph coloring, Hamiltonian cycle, satisfiability

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.163

1 Introduction

Background. Sparsification refers to the method of reducing an object such as a graph or CNF-formula to an equivalent object that is less dense, that is, an object in which the ratio of edges to vertices (or clauses to variables) is smaller. The notion is fruitful in theoretical [16] and practical (cf. [10]) settings when working with (hyper)graphs and formulas. The theory of kernelization, originating from the field of parameterized complexity theory, can be used to analyze the limits of polynomial-time sparsification. Using tools developed in the last five years, it has become possible to address questions such as: "Is there a polynomial-time algorithm that reduces an n-vertex instance of my favorite graph problem to an equivalent instance with a subquadratic number of edges?"

The impetus for this line of analysis was given by an influential paper by Dell and van Melkebeek [8] (conference version in 2010). One of their main results states that if there is an $\varepsilon > 0$ and a polynomial-time algorithm that reduces any n-vertex instance of Vertex Cover to an equivalent instance, of an arbitrary problem, that can be encoded in $\mathcal{O}(n^{2-\varepsilon})$

* This work was supported by NWO Veni grant "Frontiers in Parameterized Preprocessing" and NWO Gravity grant "Networks".

bits, then NP $\subseteq$ coNP/poly and the polynomial-time hierarchy collapses. Since any nontrivial input (G, k) of Vertex Cover has $k \leq n = |V(G)|$, their result implies that the number of edges in the $2k$-vertex kernel for k-Vertex Cover [22] cannot be improved to $\mathcal{O}(k^{2-\varepsilon})$ unless NP $\subseteq$ coNP/poly.

Using related techniques, Dell and van Melkebeek also proved important lower bounds for d-cnf-sat problems: testing the satisfiability of a propositional formula in CNF form, where each clause has at most d literals. They proved that for every fixed integer $d \geq 3$, the existence of a polynomial-time algorithm that reduces any n-variable instance of d-cnf-sat to an equivalent instance, of an arbitrary problem, with $\mathcal{O}(n^{d-\varepsilon})$ bits, for some $\varepsilon > 0$ implies NP $\subseteq$ coNP/poly. Their lower bound is tight: there are $\mathcal{O}(n^d)$ possible clauses of size d over n variables, allowing an instance to be represented by a vector of $\mathcal{O}(n^d)$ bits that specifies for each clause whether or not it is present.

Our results. We continue this line of investigation and analyze sparsification for several classic problems in graph theory and logic. We obtain several sparsification lower bounds that imply that the quadratic number of edges in existing linear-vertex kernels is likely to be unavoidable. When it comes to problems from logic, we give the—to the best of our knowledge—first example of a problem that *does* admit nontrivial sparsification: d-Not-All-Equal-SAT. We also provide a matching lower bound.

The first problem we consider is 4-Coloring, which asks whether the input graph has a proper vertex coloring with 4 colors. Using several new gadgets, we give a cross-composition [3] to show that the problem has no compression of size $\mathcal{O}(n^{2-\varepsilon})$ unless NP $\subseteq$ coNP/poly. To obtain the lower bound, we give a polynomial-time construction that embeds the logical or of a series of t size-n inputs of an NP-hard problem into a graph G' with $\mathcal{O}(\sqrt{t} \cdot n^{\mathcal{O}(1)})$ vertices, such that G' has a proper 4-coloring if and only if there is a *yes*-instance among the inputs. The main structure of the reduction follows the approach of Dell and Marx [7]: we create a table with two rows and $\mathcal{O}(\sqrt{t})$ columns and $\mathcal{O}(n^{\mathcal{O}(1)})$ vertices in each cell. For each way of picking one cell from each row, we aim to embed one instance into the edge set between the corresponding groups of vertices. When the NP-hard starting problem is chosen such that the t inputs each decompose into two induced subgraphs with a simple structure, one can create the vertex groups and their connections such that for each pair of cells (i, j), the subgraph they induce represents the $i \cdot \sqrt{t} + j$-th input. If there is a *yes*-instance among the inputs, this leads to a pair of cells that can be properly colored in a structured way. The challenging part of the reduction is to ensure that the edges in the graph corresponding to *no*-inputs do not give conflicts when extending this partial coloring to the entire graph.

The next problem we attack is Hamiltonian Cycle. We rule out compressions of size $\mathcal{O}(n^{2-\varepsilon})$ for the directed and undirected variant of the problem, under the assumption that NP $\not\subseteq$ coNP/poly. The construction is inspired by kernelization lower bounds for Directed Hamiltonian Cycle parameterized by the vertex-deletion distance to a directed graph whose underlying undirected graph is a path [2].

By combining gadgets from kernelization lower bounds for two different parameterizations of Red Blue Dominating Set, we prove that there is no compression of size $\mathcal{O}(n^{2-\varepsilon})$ for Dominating Set unless NP $\subseteq$ coNP/poly. The same construction rules out subquadratic compressions for Connected Dominating Set. These lower bounds have implications for the kernelization complexity of the parametric duals Nonblocker and Max Leaf Spanning Tree of (Connected) Dominating Set. For both Nonblocker and Max Leaf there are kernels with $\mathcal{O}(k)$ vertices [6, 11] that have $\Theta(k^2)$ edges. Our lower bounds imply that the number of edges in these kernels cannot be improved to $\mathcal{O}(k^{2-\varepsilon})$, unless NP $\subseteq$ coNP/poly.

The final family of problems we consider is d-Not-All-Equal-SAT for fixed $d \geq 4$. The input consists of a formula in CNF-form with at most d literals per clause. The question is whether there is an assignment to the variables such that each clause contains both a variable that evaluates to *true* and one that evaluates to *false*. There is a simple linear-parameter transformation from d-CNF-SAT to $(d+1)$-NAE-SAT that consists of adding one variable that occurs as a positive literal in all clauses. By the results of Dell and van Melkebeek discussed above, this implies that d-NAE-SAT does not admit compressions of size $\mathcal{O}(n^{d-1-\varepsilon})$ unless NP $\subseteq$ coNP/poly. We prove the surprising result that this lower bound is tight! A linear-algebraic result due to Lovász [21], concerning the size of critically 3-chromatic d-uniform hypergraphs, can be used to give a kernel for d-NAE-SAT with $\mathcal{O}(n^{d-1})$ clauses for every fixed d. The kernel is obtained by computing the basis of an associated matrix and removing the clauses that can be expressed as a linear combination of the basis clauses.

Related work. Dell and Marx introduced the table structure for compression lower bounds in their study of compression for packing problems [7]. Hermelin and Wu [15] analyzed similar problems. Other papers about polynomial kernelization and sparsification lower bounds include [5] and [17].

2 Preliminaries

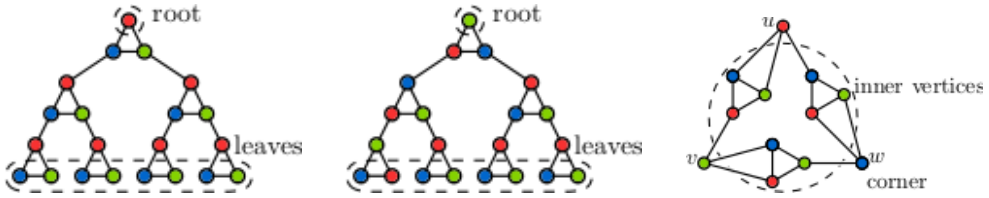
A parameterized problem $\mathcal{Q}$ is a subset of $\Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet. Let $\mathcal{Q}, \mathcal{Q}' \subseteq \Sigma^* \times \mathbb{N}$ be parameterized problems and let $h\colon \mathbb{N} \to \mathbb{N}$ be a computable function. A *generalized kernel for $\mathcal{Q}$ into $\mathcal{Q}'$ of size $h(k)$* is an algorithm that, on input $(x,k) \in \Sigma^* \times \mathbb{N}$, takes time polynomial in $|x|+k$ and outputs an instance (x',k') such that: (i) $|x'|$ and k' are bounded by $h(k)$, and (ii) $(x',k') \in \mathcal{Q}'$ if and only if $(x,k) \in \mathcal{Q}$. The algorithm is a *kernel* for $\mathcal{Q}$ if $\mathcal{Q}' = \mathcal{Q}$. It is a *polynomial (generalized) kernel* if $h(k)$ is a polynomial.

Since a polynomial-time reduction to an equivalent sparse instance yields a generalized kernel, we will use the concept of generalized kernels in the remainder of this paper to prove the non-existence of such sparsification algorithms. We employ the cross-composition framework by Bodlaender *et al.* [3], which builds on earlier work by several authors [1, 8, 13].

▶ **Definition 1** (Polynomial equivalence relation)**.** An equivalence relation $\mathcal{R}$ on Σ^* is called a *polynomial equivalence relation* if the following conditions hold. (i) There is an algorithm that, given two strings $x, y \in \Sigma^*$, decides whether x and y belong to the same equivalence class in time polynomial in $|x|+|y|$. (ii) For any finite set $S \subseteq \Sigma^*$ the equivalence relation $\mathcal{R}$ partitions the elements of S into a number of classes that is polynomially bounded in the size of the largest element of S.

▶ **Definition 2** (Cross-composition)**.** Let $L \subseteq \Sigma^*$ be a language, let $\mathcal{R}$ be a polynomial equivalence relation on Σ^*, let $\mathcal{Q} \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem, and let $f\colon \mathbb{N} \to \mathbb{N}$ be a function. An OR-*cross-composition of L into $\mathcal{Q}$* (with respect to $\mathcal{R}$) *of cost $f(t)$* is an algorithm that, given t instances $x_1, x_2, \ldots, x_t \in \Sigma^*$ of L belonging to the same equivalence class of $\mathcal{R}$, takes time polynomial in $\sum_{i=1}^{t} |x_i|$ and outputs an instance $(y,k) \in \Sigma^* \times \mathbb{N}$ such that: (i) the parameter k is bounded by $\mathcal{O}(f(t) \cdot (\max_i |x_i|)^c)$, where c is some constant independent of t, and (ii) $(y,k) \in \mathcal{Q}$ if and only if there is an $i \in [t]$ such that $x_i \in L$.

▶ **Theorem 3** ([3])**.** *Let $L \subseteq \Sigma^*$ be a language, let $\mathcal{Q} \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem, and let d, ε be positive reals. If L is NP-hard under Karp reductions, has an* OR*-cross-composition into $\mathcal{Q}$ with cost $f(t) = t^{1/d+o(1)}$, where t denotes the number of instances, and $\mathcal{Q}$ has a polynomial (generalized) kernelization with size bound $\mathcal{O}(k^{d-\varepsilon})$, then* NP $\subseteq$ coNP/poly.



(a) Treegadget with no red leaf. **(b)** Treegadget where one of the leaves is red. **(c)** Triangular gadget.

Figure 1 Used gadgets with example colorings.

For $r \in \mathbb{N}$ we will refer to an OR-cross-composition of cost $f(t) = t^{1/r} \log(t)$ as a *degree-r cross-composition*. By Theorem 2, a degree-r cross-composition can be used to rule out generalized kernels of size $\mathcal{O}(k^{r-\varepsilon})$. We frequently use the fact that a polynomial-time linear-parameter transformation from problem $\mathcal{Q}$ to $\mathcal{Q}'$ implies that any generalized kernelization lower bound for $\mathcal{Q}$, also holds for $\mathcal{Q}'$ (cf. [3, 4]). Let $[r]$ be defined as $[r] := \{x \in \mathbb{N} \mid 1 \leq x \leq r\}$. For statements marked with a $(\star)$, the proof can be found in the full version [19].

3 4-Coloring

In this section we analyze the 4-Coloring problem, which asks whether it is possible to assign each vertex of the input graph one out of 4 possible colors, such that there is no edge whose endpoints share the same color. We show that 4-Coloring does not have a generalized kernel of size $\mathcal{O}(n^{2-\varepsilon})$, by giving a degree-2 cross-composition from a tailor-made problem that will be introduced below. Before giving the construction, we first present and analyze some of the gadgets that will be needed.

▶ **Definition 4.** A *treegadget* is the graph obtained from a complete binary tree by replacing each vertex v by a triangle on vertices r_v, x_v and y_v. Let r_v be connected to the parent of v and let x_v and y_v be connected to the left and right subtree of v. An example of a treegadget with 8 leaves is shown in Figure 1. If vertex v is the root of the tree, then r_v is named the *root* of the treegadget. If v does not have a left subtree, then x_v is a *leaf* of this gadget, similarly, if v does not have a right subtree then we refer to y_v as a leaf of the gadget. Let the *height* of a treegadget be equal to the height of its corresponding binary tree.

It is easy to see that a treegadget is 3-colorable. The important property of this gadget is that if there is a color that does not appear on any leaf in a proper 3-coloring, then this must be the color of the root. See Figure 1a for an illustration.

▶ **Lemma 5.** *Let T be a treegadget with root r and let $c\colon V(T) \to \{1, 2, 3\}$ be a proper 3-coloring of T. If $k \in \{1, 2, 3\}$ such that $c(v) \neq k$ for every leaf v of T, then $c(r) = k$.*

Proof. This will be proven using induction on the structure of a treegadget. For a single triangle, the result is obvious. Suppose we are given a treegadget of height h and that the statement holds for all treegadgets of smaller height. Consider the top triangle r, x, y where r is the root. Then, by the induction hypothesis, the roots of the left and right subtree (if non-empty) are colored using k. If the left or right subtree is empty, x or y is a leaf. Hence x and y do not use color k. Since x, y, r is a triangle, r has color k in the 3-coloring. ◀

The following lemma will be used in the correctness proof of the cross-composition to argue that the existence of a single *yes*-input is sufficient for 4-colorability of the entire graph.

▶ **Lemma 6.** *Let T be a treegadget with leaves $L \subseteq V(T)$ and root r. Any 3-coloring $c' : L \to \{1,2,3\}$ that is proper on $T[L]$ can be extended to a proper 3-coloring of T. If there is a leaf $v \in L$ such that $c'(v) = i$, then such an extension exists with $c(r) \neq i$.*

Proof. We will prove this by induction on the height of the treegadget. For a single triangle, the result is obvious. Suppose the lemma is true for all treegadgets up to height $h-1$ and we are given a treegadget of height h with root triangle r, x, y and with coloring of the leaves c'. Let one of the leaves be colored using i. Without loss of generality assume this leaf is in the left subtree, which is connected to x. By the induction hypothesis, we can extend the coloring restricted to the leaves of the left subtree to a proper 3-coloring of the left subtree such that $c(r_1) \neq i$. We assign color i to x. Since c' restricted to the leaves in the right subtree is a proper 3-coloring of the leaves in the right subtree, by induction we can extend that coloring to a proper 3-coloring of the right subtree. Suppose the root of this subtree gets color $j \in \{1,2,3\}$. We now color y with a color $k \in \{1,2,3\} \setminus \{i,j\}$, which must exist. Finally, choose $c(r) \in \{1,2,3\} \setminus \{i,k\}$. By definition, the vertices r, y, and x are now assigned a different color. Both x and y have a different color than the root of their corresponding subtree, thereby c is a proper coloring. We obtain that the defined coloring c is a proper coloring extending c' with $c(r) \neq i$. ◀

▶ **Definition 7.** A *triangular gadget* is a graph on 12 vertices depicted in Figure 1c. Vertices u, v, and w are the *corners* of the gadget, all other vertices are referred to as *inner vertices*.

It is easy to see that a triangular gadget is always 3-colorable in such a way that every corner gets a different color. Moreover, we make the following observation.

▶ **Observation 8.** *Let G be a triangular gadget with corners u,v and w and let $c : V(G) \to \{1,2,3\}$ be a proper 3-coloring of G. Then $c(v) \neq c(u) \neq c(w) \neq c(v)$. Furthermore, every partial coloring that assigns distinct colors to the three corners of a triangular gadget can be extended to a proper 3-coloring of the entire gadget.*

Having presented all the gadgets we use in our construction, we now define the source problem for the cross-composition. It is a variant of the problem that was used to prove kernel lower bounds for Chromatic Number parameterized by vertex cover [3].

2-3-Coloring with Triangle Split Decomposition
Input: A graph G with a partition of its vertex set into $X \cup Y$ such that $G[X]$ is an edgeless graph and $G[Y]$ is a disjoint union of triangles.
Question: Is there a proper 3-coloring $c : V(G) \to \{1,2,3\}$ of G, such that $c(x) \in \{1,2\}$ for all $x \in X$? We will refer to such a coloring as a *2-3-coloring* of G.

▶ **Lemma 9** (⋆). 2-3-Coloring with Triangle Split Decomposition *is NP-complete.*

▶ **Theorem 10.** 4-Coloring *parameterized by the number of vertices n does not have a generalized kernel of size $\mathcal{O}(n^{2-\varepsilon})$ for any $\varepsilon > 0$, unless* NP $\subseteq$ coNP/poly.

Proof. By Theorem 3 and Lemma 9 it suffices to give a degree-2 cross-composition from the 2-3-coloring problem defined above into 4-Coloring parameterized by the number of vertices. For ease of presentation, we will actually give a cross-composition into the 4-List Coloring problem, whose input consists of a graph G and a list function that assigns every vertex $v \in V(G)$ a list $L(v) \subseteq [4]$ of allowed colors. The question is whether there is a proper coloring of the graph in which every vertex is assigned a color from its list. The 4-List Coloring reduces to the ordinary 4-Coloring by a simple transformation that adds a

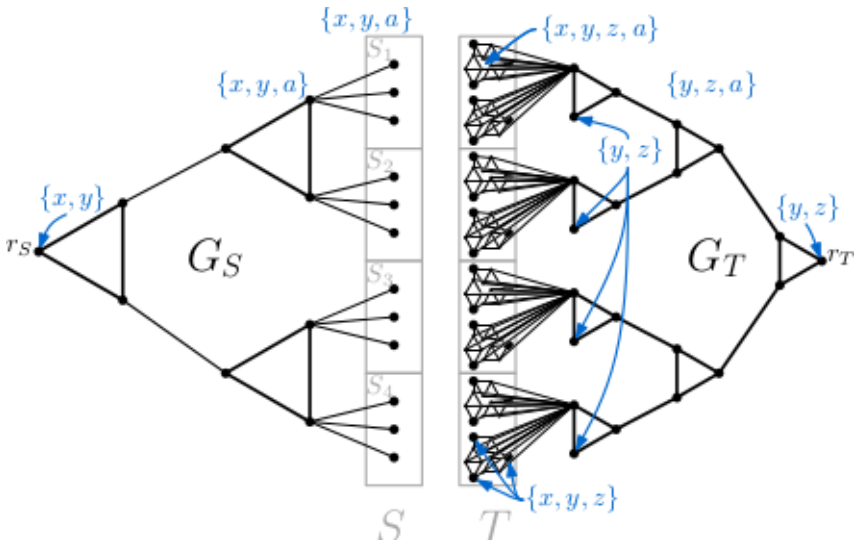


Figure 2 Graph G' for $t' = 4$, $m = 3$ and $n = 2$. Edges between vertices in S and T are left out.

4-clique to enforce the color lists, which will prove the theorem. For now, we focus on giving a cross-composition into 4-List Coloring.

We start by defining a polynomial equivalence relation on inputs of 2-3-Coloring with Triangle Split Decomposition. Let two instances of 2-3-Coloring with Triangle Split Decomposition be equivalent under equivalence relation $\mathcal{R}$ when they have the same number of triangles and the independent sets also have the same size. It is easy to see that $\mathcal{R}$ is a polynomial equivalence relation. By duplicating one of the inputs, we can ensure that the number of inputs to the cross-composition is an even power of two; this does not change the value of or, and increases the total input size by at most a factor four. We will therefore assume that the input consists of t instances of 2-3-Coloring with Triangle Split Decomposition such that $t = 2^{2i}$ for some integer i, implying that $\sqrt{t}$ and $\log \sqrt{t}$ are integers. Let $t' := \sqrt{t}$. Enumerate the instances as $X_{i,j}$ for $1 \leq i, j \leq t'$. Each input $X_{i,j}$ consists of a graph $G_{i,j}$ and a partition of its vertex set into sets U and V, such that U is an independent set of size m and $G_{i,j}[V]$ consists of n vertex-disjoint triangles. Enumerate the vertices in U and V as $u_1, \ldots, u_m$ and $v_1, \ldots, v_{3n}$, such that vertices $v_{3\ell-2}, v_{3\ell-1}$ and $v_{3\ell}$ form a triangle, for $\ell \in [n]$. We will create an instance G' of the 4-List-Coloring problem, which consists of a graph G' and a list function L that assigns each vertex a subset of the color palette $\{x, y, z, a\}$. Refer to Figure 2 for a sketch of G'.

1. Initialize G' as the graph containing t' sets of m vertices each, called S_i for $i \in [t']$. Label the vertices in each of these sets as s^i_ℓ for $i \in [t']$, $\ell \in [m]$ and let $L(s^i_\ell) := \{x, y, a\}$.
2. Add t' sets of n triangular gadgets each, labeled T_j for $j \in [t']$. Label the corner vertices in T_j as t^j_ℓ for $\ell \in [3n]$, such that vertices $t^j_{3\ell-2}, t^j_{3\ell-1}$ and $t^j_{3\ell}$ are the corner vertices of one of the gadgets for $\ell \in [n]$. Let $L(t^j_\ell) := \{x, y, z\}$ and for any inner vertex v of a triangular gadget, let $L(v) := \{x, y, z, a\}$.
3. Connect vertex s^i_k to vertex t^j_ℓ if in graph $G_{i,j}$ vertex u_k is connected to v_ℓ, for $k \in [m]$ and $\ell \in [3n]$. By this construction, the subgraph of G' induced by $S_i \cup T_j$ is isomorphic to the graph obtained from $G_{i,j}$ by replacing each triangle with a triangular gadget.
4. Add a treegadget G_S with t' leaves to G' and enumerate these leaves as $1, \ldots, t'$; recall that t' is a power of two. Connect the i'th leaf of G_S to every vertex in S_i. Let the root of G_S be r_S and define $L(r_S) := \{x, y\}$. For every other vertex v in G_S let $L(v) := \{x, y, a\}$.

5. Add a treegadget G_T with $2t'$ leaves to G' and enumerate these leaves as $1, \ldots, 2t'$. For $j \in [t']$, connect every inner vertex of a triangular gadget in group T_j to leaf number $2j-1$ of G_T. For every leaf v with an even index let $L(v) := \{y, z\}$ and let the root r_T have list $L(r_T) := \{y, z\}$. For every other vertex v of gadget G_T let $L(v) := \{y, z, a\}$.

▶ **Claim 11.** *The graph G' is 4-list-colorable $\Leftrightarrow$ some input instance $X_{i^*j^*}$ is 2-3-colorable.*

Proof. ($\Rightarrow$) Suppose we are given a 4-list coloring c for G'. By definition, $c(r_S) \neq a$. From Lemma 5 it follows that there is a leaf v of G_S such that $c(v) = a$. This leaf is connected to all vertices in some S_{i^*}, which implies that none of the vertices in S_{i^*} are colored using a. Therefore all vertices in S_{i^*} are colored using x and y. Similarly the gadget G_T has at least one leaf v such that $c(v) = a$, note that this must be a leaf with an odd index. Therefore there exists T_{j^*} where all vertices are colored using x,y or z. Thereby in $S_{i^*} \cup T_{j^*}$ only three colors are used, such that S_{i^*} is colored using only two colors. Using Observation 8 and the fact that $G'[S_{i^*} \cup T_{j^*}]$ is isomorphic to the graph obtained from G_{i^*,j^*} by replacing triangles by triangular gadgets, we conclude that $X_{i^*j^*}$ has a proper 2-3-coloring.

($\Leftarrow$) Suppose $c\colon V(G_{i^*,j^*}) \to \{x, y, z\}$ is a proper 2-3-coloring for X_{i^*,j^*}. We will construct a 4-list coloring $c'\colon V(G') \to \{x, y, z, a\}$ for G'. For u_k, $k \in [m]$ in instance X_{i^*,j^*} let $c'(s_k^{i^*}) := c(u_k)$ and for v_ℓ for $\ell \in [3n]$ let $c'(t_\ell^{j^*}) := c(v_\ell)$. Let $c'(s_\ell^i) := a$ for $i \neq i^*$ and $\ell \in [n]$, furthermore let $c'(t_\ell^j) := z$ for $j \neq j^*$ and $\ell \in [3m]$. For triangular gadgets in T_{j^*} the coloring c' defines all corners to have distinct colors; by Observation 8 we can color the inner vertices consistently using $\{x, y, z\}$. For T_j with $j \in [t']$ and $j \neq j^*$, the corners of triangular gadgets have color z and we can now consistently color the inner vertices using $\{x, y, a\}$.

The leaf of gadget G_S that is connected to S_{i^*} can be colored using a. Every other leaf can use both x and y, so we can properly 3-color the leaves such that one leaf has color a. From Lemma 6 it follows that we can consistently 3-color G_S such that the root r_S does not receive color a, as required by $L(r_S)$. Similarly, in triangular gadgets in T_{j^*} the inner vertices do not have color a. As such, leaf $2j^* - 1$ of G_T can be colored using a and we color leaf $2j^*$ with y. For $j \in [t']$ with $j \neq j^*$ color leaf $2j-1$ with z and leaf $2j$ using y. Now the leaves of G_T are properly 3-colored and one is colored a. It follows from Observation 8 that we can color G_T such that the root is not colored a. This completes the 4-list coloring of G'. ◀

The claim shows that the construction serves as a cross-composition into 4-List Coloring. To prove the theorem, we add four new vertices to simulate the list function. Add a clique on 4 vertices $\{x, y, z, a\}$. If for any vertex v in G', some color is not contained in $L(v)$, connect v to the vertex corresponding to this color. As proper colorings of the resulting graph correspond to proper list colorings of G', the resulting graph is 4-colorable if and only if there is a *yes*-instance among the inputs. It remains to bound the parameter of the problem, i.e., the number of vertices. Observe that a treegadget has at least as many leaves as its corresponding binary tree, therefore the graph G' has at most $12mt' + nt' + 6t' + 12t' + 4 = \mathcal{O}(t' \cdot (m+n)) = \mathcal{O}(\sqrt{t} \max |X_{i,j}|)$ vertices. Theorem 10 now follows from Theorem 3 and Lemma 9. ◀

4 Hamiltonian cycle

In this section we prove a sparsification lower bound for Hamiltonian Cycle and its directed variant. The starting problem is Hamiltonian $s-t$ path on bipartite graphs.

HAMILTONIAN $s-t$ PATH ON BIPARTITE GRAPHS
Input: An undirected bipartite graph G with partite sets A and B such that $|B| = n = |A| + 1$, together with two distinguished vertices b_1 and b_n that have degree 1.
Question: Does G have a Hamiltonian path from b_1 to b_n?

It is known that Hamiltonian path is NP-complete on bipartite graphs [14] and it is easy to see that is remains NP-complete when fixing a degree 1 start and endpoint.

▶ **Theorem 12.** (DIRECTED) HAMILTONIAN CYCLE *parameterized by the number of vertices n does not have a generalized kernel of size $\mathcal{O}(n^{2-\varepsilon})$ for any $\varepsilon > 0$, unless* $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

Proof. By a suitable choice of polynomial equivalence relation, and by padding the number of inputs, it suffices to give a cross-composition from the $s-t$ problem on bipartite graphs when the input consists of t instances $X_{i,j}$ for $i, j \in [\sqrt{t}]$ (i.e., $\sqrt{t}$ is an integer), such that each instance $X_{i,j}$ encodes a bipartite graph $G_{i,j}$ with partite sets $A^*_{i,j}$ and $B^*_{i,j}$ with $|A^*_{i,j}| = m$ and $|B^*_{i,j}| = n = m + 1$, for some $m \in \mathbb{N}$. For each instance, label all elements in $A^*_{i,j}$ as $a^*_1, \ldots, a^*_m$ and all elements in $B^*_{i,j}$ as $b^*_1, \ldots, b^*_n$ such that b^*_1 and b^*_n have degree 1.

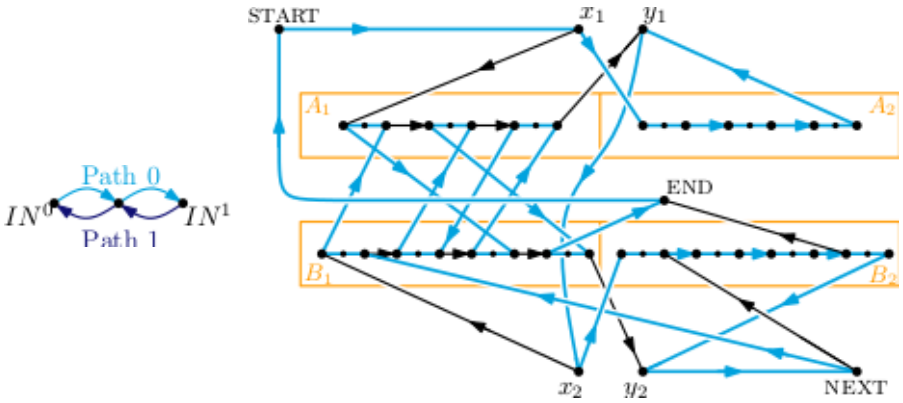
The construction makes extensive use of the path gadget depicted in Figure 3a. Observe that if G' contains a path gadget as an induced subgraph, while the remainder of the graph only connects to its terminals IN^0 and IN^1, then any Hamiltonian cycle in G' traverses the path gadget in one of the two ways depicted in Figure 3a. We create an instance G' of DIRECTED HAMILTONIAN CYCLE that acts as the logical OR of the inputs.

1. First of all construct $\sqrt{t}$ groups of m path gadgets each. Refer to these groups as A_i, for $i \in [\sqrt{t}]$, and label the gadgets within group A_i as $a^i_1, \ldots, a^i_m$. Let the union of all created sets A_i be named A. Similarly, construct $\sqrt{t}$ groups of n path gadgets each. Refer to these groups as B_j, for $j \in [\sqrt{t}]$, and label the gadgets within group B_j as $b^j_1, \ldots, b^j_n$. Let B be the union of all B_j for $j \in [\sqrt{t}]$.
2. For every input instance $X_{i,j}$, for each edge $\{a^*_k, b^*_\ell\}$ in $X_{i,j}$ with $k \in [m]$, $\ell \in [n]$, add an arc from IN^0 of a^i_k to IN^1 of b^j_ℓ and an arc from IN^0 of b^j_ℓ to IN^1 of a^i_k.

If some $X_{i,j}$ has a Hamiltonian $s-t$ path, it can be mimicked by the combination of A_i and B_j, where for each vertex in $X_{i,j}$ we traverse its path gadget in G', following Path 1. The following construction steps are needed to extend such a path to a Hamiltonian cycle in G'.

3. Add an arc from the IN^1 terminal of a^i_ℓ to the IN^0 terminal of $a^i_{\ell+1}$ for all $\ell \in [m-1]$ and all $i \in [\sqrt{t}]$. Similarly add an arc from the IN^1 terminal of b^i_ℓ to the IN^0 of $b^i_{\ell+1}$ for all $\ell \in [n-1]$ and all $i \in [\sqrt{t}]$.
4. Add a vertex START and a vertex END and the arc (END, START).
5. Let $r := \sqrt{t} - 1$, add $2r$ tuples of vertices, x_i, y_i for $i \in [2r]$ and connect START to x_1. Furthermore, add the arcs (y_i, x_{i+1}) for $i \in [2r-1]$.
6. For $i \leq r$ we add arcs from x_i to the IN^0 terminal of the gadgets $a^j_1, j \in [\sqrt{t}]$. Furthermore we add an arc from IN^1 of a^j_m to y_i for all $j \in [\sqrt{t}]$ and $i \in [r]$. When $i > r$ add arcs from x_i to the IN^0 terminal of b^j_1 for $j \in [\sqrt{t}]$ and connect IN^1 of b^j_n to y_i.
7. Add a vertex NEXT and the arc (y_{2r}, NEXT) and an arc from NEXT to the IN^1 terminal of all gadgets b^j_1 for $j \in [\sqrt{t}]$.
8. Furthermore, add arcs from IN^0 of all gadgets b^j_n to END for $j \in [\sqrt{t}]$. So for each B_j, exactly one vertex has an outgoing arc to END and one has an incoming arc from NEXT.

This completes the construction of G'. A sketch of G' is shown in Figure 3b.



(a) A path gadget. **(b)** The general structure of the created graph, when given 4 inputs with $n = 3$ and $m = 4$.

Figure 3 Illustrations for the lower bound for Hamiltonian Cycle.

▶ **Lemma 13** (⋆). *Graph G' has a directed Hamiltonian cycle if and only if at least one of the instances $X_{i,j}$ has a Hamiltonian $s-t$-path.*

The number of vertices of G' is $3(m+n)\sqrt{t}+3\cdot 2(\sqrt{t}-1)+3 = \mathcal{O}(\sqrt{t}\cdot(m+n)) = \mathcal{O}(\sqrt{t}\cdot\max|X_{i,j}|)$. By with Lemma 13 the construction is a degree-2 cross-composition from Hamiltonian $s-t$-paths in Bipartite graphs to Directed Hamiltonian cycle parameterized by the number of vertices, proving the generalized kernel lower bound for the directed problem. Karp [20] gave a polynomial-time reduction that, given an n-vertex directed graph G, produces an undirected graph G' with $3n$ vertices such that G has a directed Hamiltonian cycle if and only if G' has a Hamiltonian cycle. This is a linear parameter transformation from directed Hamiltonian cycle to Hamiltonian cycle. Since linear-parameter transformations transfer lower bounds [3, 4], we conclude that (Directed) Hamiltonian cycle does not have a generalized kernel of size $\mathcal{O}(n^{2-\varepsilon})$ for any $\varepsilon > 0$. ◀

5 Dominating set

In this section we discuss the Dominating Set problem and its variants. Dom *et al.* [9] proved several kernelization lower bounds for the variant Red-Blue Dominating Set, which is the variant on bipartite (red/blue colored) graphs in which the goal is to dominate all the blue vertices by selecting a small subset of red vertices. Using ideas from their kernel lower bounds for the parameterization by either the number of red or the number of blue vertices, we prove sparsification lower bounds for (Connected) Dominating Set. Since we parameterize by the number of vertices, the same lower bounds apply to the dual problems Nonblocker and Max Leaf Spanning Tree, resulting in the following theorem.

▶ **Theorem 14** (⋆). (Connected) Dominating Set, Nonblocker, *and* Max Leaf Spanning Tree *parameterized by the number of vertices n do not have a generalized kernel of size $\mathcal{O}(n^{2-\varepsilon})$ for any $\varepsilon > 0$, unless* NP $\subseteq$ coNP/poly.

The proof is deferred to the complete version [19] due to space restrictions. Just as the sparsification lower bounds for Vertex Cover that were presented by Dell and van Melkebeek [8] had implications for the parameterization by the solution size k, Theorem 14 has implications for the kernelization complexity of k-Nonblocker and k-Max Leaf.

Since the solution size k never exceeds the number of vertices in this problem, a kernel with $\mathcal{O}(k^{2-\epsilon})$ edges would give a nontrivial sparsification, contradicting Theorem 14. Hence our results show that the existing linear-vertex kernels for k-Nonblocker [6] and k-Max Leaf [11] cannot be improved to $\mathcal{O}(k^{2-\varepsilon})$ edges unless NP $\subseteq$ coNP/poly.

6 d-Hypergraph 2-Colorability and d-NAE-SAT

The goal of this section is to give a nontrivial sparsification algorithm for NAE-SAT and prove a matching lower bound. For ease of presentation, we start by analyzing the closely related hypergraph 2-colorability problem. Recall that a hypergraph consists of a vertex set V and a set E of *hyperedges*; each hyperedge $e \in E$ is a subset of V. A 2-coloring of a hypergraph is a function $c\colon V \to \{1,2\}$; such a coloring is *proper* if there is no hyperedge whose vertices all obtain the same color. We will use d-Hypergraph 2-Colorability to refer to the setting where hyperedges have size at most d. The corresponding decision problem asks, given a hypergraph, whether it is 2-colorable.

▶ **Theorem 15.** *d-Hypergraph 2-Colorability parameterized by the number of vertices n has a kernel with $2 \cdot n^{d-1}$ hyperedges that can be encoded in $\mathcal{O}(n^{d-1} \cdot d \cdot \log n)$ bits.*

Proof. Suppose we are given a hypergraph with vertex set V and hyperedges E, where each hyperedge contains at most d vertices. We show how to reduce the number of hyperedges without changing the 2-colorability status. Let $E_r \subseteq E$ denote the set of edges in E that contain exactly r vertices. For each E_r we construct a set $E'_r \subseteq E_r$ of *representative hyperedges*. Enumerate the edges in E_r as $e^r_1, \ldots, e^r_k$. We construct a $(0,1)$-matrix M_r with $N := \binom{n}{r-1}$ rows and k columns. Consider all possible subsets $A_1, \ldots, A_N$ of size $r-1$ of the set of vertices V. Define the elements $m_{i,j}$ for $i \in N$ and $j \in k$ of M_r as follows.

$$m_{i,j} := \begin{cases} 1 & \text{if } A_i \subseteq e^r_j; \\ 0 & \text{otherwise.} \end{cases}$$

Using Gaussian elimination, compute a basis B of the columns of this matrix, which is a subset of the columns that span the column space of M_r. Let E'_r contain edge e^r_i if the i'th column of M_r is contained in B, and define $E' := \bigcup_{r \in [d]} E'_r$, which forms the kernel. Using a lemma due to Lovász [21], we can prove that E' preserves the 2-colorability status.

▶ **Lemma 16** ($\star$). *(V,E) has a proper 2-coloring $\Leftrightarrow$ (V,E') has a proper 2-coloring.*

To bound the size of the kernel, consider the matrix M_r for $r \in [d]$. Its rank is bounded by the minimum of its number of rows and columns, which is at most $\binom{n}{r-1} \leq n^{r-1}$. As such, we get $|E'_r| \leq \operatorname{rank}(M_r) \leq n^{r-1}$. Note that $d \leq n$, such that $|E'| \leq \sum_{r=1}^{d} n^{r-1} = n^{d-1} + \sum_{r=1}^{d-1} n^{r-1} \leq 2 \cdot n^{d-1}$. So E' contains at most $2n^{d-1}$ hyperedges. Since a hyperedge consists of at most d vertices, the kernel can be encoded in $\mathcal{O}(n^{d-1} \cdot d \cdot \log n)$ bits. ◀

By a folklore reduction, Theorem 15 gives a sparsification for NAE-SAT. Consider an instance of d-NAE-SAT, which is a conjunction of clauses of size at most d over variables $x_1, \ldots, x_n$. The formula gives rise to a hypergraph on vertex set $\{x_i, \neg x_i \mid i \in [n]\}$ containing one hyperedge per clause, whose vertices correspond to the literals in the clause. When additionally adding n hyperedges $\{x_i, \neg x_i\}$ for $i \in [n]$, it is easy to see that the resulting hypergraph is 2-colorable if and only if there is a NAE-satisfying assignment to the formula. The maximum size of a hyperedge matches the maximum size of a clause and the number of created vertices is twice the number of variables. We can therefore sparsify an n-variable

instance of d-NAE-SAT in the following way: reduce it to a d-hypergraph with $n' := 2n$ vertices and apply the kernelization algorithm of Theorem 15. It is easy to verify that restricting the formula to the representative hyperedges in the kernel gives an equisatisfiable formula containing $2 \cdot (n')^{d-1} \in \mathcal{O}(2^{d-1}n^{d-1})$ clauses, giving a sparsification for NAE-SAT. As mentioned in the introduction, the existence of a linear-parameter transformation [18] from d-CNF-SAT to $(d+1)$-NAE-SAT also implies a sparsification *lower bound* for d-NAE-SAT, using the results of Dell and van Melkebeek [8]. Hence we obtain the following theorem.

▶ **Theorem 17.** *For every fixed $d \geq 4$, the d-NAE-SAT problem parameterized by the number of variables n has a kernel with $\mathcal{O}(n^{d-1})$ clauses that can be encoded in $\mathcal{O}(n^{d-1} \cdot \log n)$ bits, but admits no generalized kernel of size $\mathcal{O}(n^{d-1-\varepsilon})$ for $\varepsilon > 0$ unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.*

7 Conclusion

We have added several classic graph problems to a growing list of problems for which non-trivial polynomial-time sparsification is provably impossible under the assumption that $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$. Our results for (Connected) Dominating Set proved that the linear-vertex kernels with $\Theta(k^2)$ edges for k-Nonblocker and k-Max Leaf Spanning Tree cannot be improved to $\mathcal{O}(k^{2-\varepsilon})$ edges unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

The graph problems for which we proved sparsification lower bounds can be defined in terms of vertices: the 4-Coloring problem asks for a partition of the vertex set into four independent sets, Dominating Set asks for a dominating subset of vertices, and Hamiltonian Cycle asks for a permutation of the vertices that forms a cycle. In contrast, not much is known concerning sparsification lower bounds for problems whose solution is an edge subset of possibly quadratic size. For example, no sparsification lower bounds are known for well-studied problems such as Max Cut, Cluster Editing, or Feedback Arc Set in Tournaments. Difficulties arise when attempting to mimic our lower bound constructions for such edge-based problems. Our constructions all embed t instances into a $2 \times \sqrt{t}$ table, using each combination of a cell in the top row and bottom row to embed one input. For problems defined in terms of edge subsets, it becomes difficult to "turn off" the contribution of edges that are incident on vertices that do not belong to the two cells that correspond to a *yes*-instance among the inputs to the or-construction. This could be interpreted as evidence that edge-based problems such as Max Cut might admit non-trivial polynomial sparsification. We have not been able to answer this question in either direction, and leave it as an open problem. For completeness, we point out that Karp's reduction [20] from Vertex Cover to Feedback Arc Set (which only doubles the number of vertices) implies, using existing bounds for Vertex Cover [8], that Feedback Arc Set does not have a compression of size $\mathcal{O}(n^{2-\varepsilon})$ unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

Another problem whose compression remains elusive is 3-Coloring. In several settings (cf. [12]), the optimal kernel size matches the size of minimal obstructions in a problem-specific partial order. This is the case for d-NAE-SAT, whose kernel with $\mathcal{O}(n^{d-1})$ clauses matches the fact that critically 3-chromatic d-uniform hypergraphs have at most $\mathcal{O}(n^{d-1})$ hyperedges. Following this line of reasoning, it is tempting to conjecture that 3-Coloring does not admit subquadratic compressions: there are critically 4-chromatic graphs with $\Theta(n^2)$ edges [23].

The kernel we have given for d-NAE-SAT is one of the first examples of non-trivial polynomial-time sparsification for general structures that are not planar or similarly guaranteed to be sparse. Obtaining non-trivial sparsification algorithms for other problems is an interesting challenge for future work. Are there natural problems defined on general graphs that admit subquadratic sparsification?

References

1 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009.

2 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernel bounds for path and cycle problems. *Theor. Comput. Sci.*, 511:117–136, 2013.

3 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM J. Discrete Math.*, 28(1):277–305, 2014.

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.

5 Marek Cygan, Fabrizio Grandoni, and Danny Hermelin. Tight kernel bounds for problems on graphs with small degeneracy. In *Proc. 21st ESA*, pages 361–372, 2013.

6 Frank K. H. A. Dehne, Michael R. Fellows, Henning Fernau, Elena Prieto, and Frances A. Rosamond. NONBLOCKER: parameterized algorithmics for minimum dominating set. In *Proc. 32nd SOFSEM*, pages 237–245, 2006.

7 Holger Dell and Dániel Marx. Kernelization of packing problems. In *Proc. 23rd SODA*, pages 68–81, 2012.

8 Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. *J. ACM*, 61(4):23:1–23:27, 2014.

9 Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Kernelization lower bounds through colors and IDs. *ACM Trans. Algorithms*, 11(2):13:1–13:20, 2014.

10 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. Sparsification - a technique for speeding up dynamic graph algorithms. *J. ACM*, 44(5):669–696, 1997.

11 Vladimir Estivill-Castro, Michael Fellows, Michael Langston, and Frances Rosamond. FPT is P-time extremal structure I. In *Proc. 1st ACiD*, pages 1–41, 2005.

12 Michael R. Fellows and Bart M. P. Jansen. FPT is characterized by useful obstruction sets: Connecting algorithms, kernels, and quasi-orders. *TOCT*, 6(4):16, 2014.

13 Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *J. Comput. Syst. Sci.*, 77(1):91–106, 2011.

14 Michael R. Garey and David S. Johnson. *Computers and Intractability.* W.H. Freeman, 1979.

15 Danny Hermelin and Xi Wu. Weak compositions and their applications to polynomial lower bounds for kernelization. In *Proc. 23rd SODA*, pages 104–113, 2012.

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

17 Bart M. P. Jansen. On sparsification for computing treewidth. *Algorithmica*, 71(3):605–635, 2015.

18 Bart M. P. Jansen and Stefan Kratsch. Data reduction for graph coloring problems. *Information and Computation*, 231:70–88, 2013.

19 Bart M. P. Jansen and Astrid Pieterse. Sparsification upper and lower bounds for graphs problems and not-all-equal SAT. *CoRR*, abs/1509.07437, 2015.

20 Richard M. Karp. Reducibility Among Combinatorial Problems. In *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

21 Lásló Lovász. Chromatic number of hypergraphs and linear algebra. In *Studia Scientiarum Mathematicarum Hungarica 11*, pages 113–114, 1976.

22 George L. Nemhauser and Leslie E. Trotter Jr. Vertex packings: structural properties and algorithms. *Math. Program.*, 8:232–248, 1975.

23 Bjarne Toft. On the maximal number of edges of critical k-chromatic graphs. *Studia Scientiarum Mathematicarum Hungarica*, 5:461–470, 1970.

Definability Equals Recognizability for k-Outerplanar Graphs*†

Lars Jaffke[1] and Hans L. Bodlaender[2]

1 **CWI Amsterdam**
Postbus 94079, 1090 GB Amsterdam, The Netherlands
l.jaffke@cwi.nl

2 **Department of Information and Computing Sciences,**
Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
Department of Mathematics and Computer Science;
University of Technology Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands
h.l.bodlaender@uu.nl

Abstract

One of the most famous algorithmic meta-theorems states that every graph property that can be defined by a sentence in counting monadic second order logic (CMSOL) can be checked in linear time for graphs of bounded treewidth, which is known as Courcelle's Theorem [6]. These algorithms are constructed as finite state tree automata, and hence every CMSOL-definable graph property is recognizable. Courcelle also conjectured that the converse holds, i.e. every recognizable graph property is definable in CMSOL for graphs of bounded treewidth. We prove this conjecture for k-outerplanar graphs, which are known to have treewidth at most $3k - 1$ [2].

1998 ACM Subject Classification F.1.1 [Theory of Computation] Models of Computation – Automata, F.4.1 [Mathematical Logic and Formal Languages] Mathematical Logic – Computational logic, G.2.2 [Discrete Mathematics] Graph Theory – Graph algorithms

Keywords and phrases treewidth, monadic second order logic of graphs, finite state tree automata, k-outerplanar graphs

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.175

1 Introduction

A seminal result from 1990 by Courcelle states that for every graph property P that can be formulated in a language called counting monadic second order logic (CMSOL), and each fixed k, there is a linear time algorithm that decides P for a graph given a tree decomposition of width at most k [6] (while similar results were discovered by Arnborg et al. [1] and Borie et al. [4]). Counting monadic second order logic generalizes monadic second order logic (MSOL) with a collection of predicates testing the size of sets modulo constants. Courcelle showed that this makes the logic strictly more powerful [6]. The algorithms constructed in Courcelle's proof have the shape of a finite state tree automaton and hence we can say that CMSOL-definable graph properties are recognizable (or, equivalently, regular or finite-state). Courcelle's Theorem generalizes one direction of a classic result in automata theory by

* The research was done while the first author was a student at Utrecht University.

† The research of the second author was partially funded by the Networks programme, funded by the Dutch Ministry of Education, Culture and Science through the Netherlands Organisation for Scientific Research.

10th International Symposium on Parameterized and Exact Computation (IPEC 2015).
Editors: Thore Husfeldt and Iyad Kanj; pp. 175–186

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

Büchi, which states that a language is recognizable, if and only if it is MSOL-definable [5]. Courcelle conjectured in 1990 that the other direction of Büchi's result can also be generalized for graphs of bounded treewidth in CMSOL, i.e. that each recognizable graph property is CMSOL-definable.

This conjecture is still regarded to be open. Its claimed resolution by Lapoire [18] is not considered to be valid by several experts. In the course of time proofs were given for the classes of trees and forests [6], partial 2-trees [7], partial 3-trees and k-connected partial k-trees [16]. A sketch of a proof for graphs of pathwidth at most k appeared at ICALP 1997 [15]. Very recently, one of the authors proved, in collaboration with Heggernes and Telle, that Courcelle's Conjecture holds for partial k-trees without chordless cycles of length at least ℓ [3].

By the results presented in this paper, we add the class of k-outerplanar graphs to this list. In particular, we first prove the conjecture for 3-connected k-outerplanar graphs and then generalize this result to all k-outerplanar graphs, based on the decomposition of a connected graph into its 3-connected components, discovered by Tutte [19] and shown to be definable in monadic second order logic by Courcelle [10].

The rest of the paper is organized as follows. In Section 2 we give the basic definitions and review the concepts involved in our proofs. We present the main result in Section 3 and conclude in Section 4.

For details of the proofs of the results given in this text, the reader is referred to the full version of the paper [14].

2 Preliminaries

Graphs and Tree Decompositions

Throughout the paper, a graph $G = (V, E)$ with vertex set V and edge set E is undirected, connected and simple. We denote the subgraph relation by $G \sqsubseteq H$ and for a set $W \subseteq V$, $G[W]$ denotes the induced subgraph over W in G, so $G[W] = (W, E \cap (W \times W))$. We call a set $C \subset V$ a *cut* of G, if $G[V \setminus C]$ is disconnected. An *ℓ-cut* of G is a cut of size ℓ. A set $S \subseteq V$ is said to be *incident* to an ℓ-cut C, if $C \subset S$. We call a graph *ℓ-connected*, if it does not contain a cut of size at most $\ell - 1$.

We now define the class of k-outerplanar graphs and some central notions used extensively throughout the rest of the paper.

▶ **Definition 1** ((Planar) Embedding)**.** A drawing of a graph in the plane is called an *embedding*. If no pair of edges in this drawing crosses, then it is called *planar*.

▶ **Definition 2** (k-Outerplanar Graph)**.** Let $G = (V, E)$ be a graph. G is called a *planar graph*, if there exists a planar embedding of G. An embedding of a graph G is 1-*outerplanar*, if it is planar, and all vertices lie on the exterior face. For $k \geq 2$, an embedding of a graph G is k-*outerplanar*, if it is planar, and when all vertices on the outer face are deleted, then one obtains a $(k-1)$-outerplanar embedding of the resulting graph. If G admits a k-outerplanar embedding, then it is called a k-*outerplanar graph*.

▶ **Definition 3** (Fundamental Cycle)**.** Let $G = (V, E)$ be a graph with maximal spanning forest $T = (V, F)$. Given an edge $e = \{v, w\}$, $e \in E \setminus F$, its *fundamental cycle* is the cycle which is formed by the unique path from v to w in F together with the edge e.

▶ **Definition 4** (Tree Decomposition, Treewidth)**.** A *tree decomposition* of a graph $G = (V, E)$ is a pair (T, X) of a tree $T = (N, F)$ and an indexed family of vertex sets $(X_t)_{t \in N}$ (called *bags*), such that the following properties hold.

(i) Each vertex $v \in V$ is contained in at least one bag.
(ii) For each edge $e \in E$ there exists a bag containing both endpoints.
(iii) For each vertex $v \in V$, the bags in the tree decomposition that contain v form a subtree of T.

The *width* of a tree decomposition is the size of the largest bag minus 1 and the *treewidth* of a graph is the minimum width of all its tree decompositions. We might sometimes refer to graphs of treewidth at most k as *partial k-trees*.[1]

To avoid confusion, in the following we will refer to elements of N as *nodes* and elements of V as *vertices*. Sometimes, to shorten the notation, we might not differ between the terms *node* and *bag* in a tree decomposition.

We use the following notation. If P denotes a graph property (e.g. a graph contains a Hamiltonian cycle), then by '$P(G)$' we express that a graph G has property P.

Monadic Second Order Logic of Graphs

We now define counting monadic second order logic of graphs $G = (V, E)$, using terminology from [4] and [16]. Variables in this predicate logic are either single vertices/edges or vertex/edge sets. We form predicates by joining *atomic predicates* (vertex equality $v = w$, vertex membership $v \in V$, edge membership $e \in E$ and vertex-edge incidence $\mathrm{Inc}(v, e)$) via negation $\neg$, conjunction $\wedge$, disjunction $\vee$, implication $\rightarrow$ and equivalence $\leftrightarrow$ together with existential quantification $\exists$ and universal quantification $\forall$ over variables in our domain $V \cup E$. To extend this monadic second order logic (MSOL) to *counting* monadic second order logic (CMSOL), one additionally allows the use of predicates $\mathrm{mod}_{p,q}(S)$ for sets S, which are true, if and only if $|S| \bmod q = p$, for constants p and q (with $p < q$).

Let ϕ denote a predicate without unquantified (so-called *free*) variables constructed as explained above and G be a graph. We call ϕ a *sentence* and denote by $G \models \phi$ that ϕ yields a truth assignment when evaluated with the graph G.

▶ **Definition 5** (Definable Graph Properties)**.** Let P denote a graph property. We say that P is *(C)MSOL-definable*, if there exists a (C)MSOL-sentence ϕ_P such that

$$P(G) \Leftrightarrow G \models \phi_P.$$

We distinguish between two types of free variables. Consider a predicate ϕ with free variables $x_1, \ldots, x_p$. A subset of $x_1, \ldots, x_p$, say $x_1, \ldots, x_a$ (where $a \leq p$), can be considered its *arguments*, and the variables $x_{a+1}, \ldots, x_p$ are its *parameters*. We denote this predicate as $\phi(x_1, \ldots, x_a)$, i.e. its parameters do not appear in the notation. We illustrate the difference between arguments and parameters in the following example.

▶ **Example 6.** Let P denote the property that a graph has a k-coloring and $\phi_{col}(v, w)$ a predicate, which is true, if and only if a vertex v has a lower numbered color than w in a given coloring. Then ϕ_{col} has two arguments, vertices v and w, and k parameters, the k color classes. Clearly, the choice of the parameters influences the evaluation of ϕ_{col}, but in most applications of parameters for predicates, it is sufficient to show that one can guess *some* variables of the evaluation graph to define a property.

[1] For several characterizations of graphs of treewidth at most k, see e.g. [2, Theorem 1]

▶ **Definition 7** ((Existentially) Definable Relations)**.** Let $R(x_1, \ldots, x_r)$ denote a relation with arguments $x_1, \ldots, x_r$. We say that R is *(C)MSOL-definable*, if there exists a parameter-free predicate $\phi_R(x_1, \ldots, x_r)$, encoding the relation R. Furthermore we call R *existentially (C)MSOL-definable*, if there exists a predicate $\phi_R(x_1, \ldots, x_r)$ with parameters $x_1, \ldots, x_p$, which, after substituting the parameters by fixed values in the evaluation graph, encodes the relation R.

A central concept used in this paper is an implicit representation of tree decompositions in monadic second order logic, as we cannot refer to its bags and edges as variables in MSOL directly. We have to define predicates, which encode the construction of a tree decomposition of each member of a given graph class. We require two types of predicates. The **Bag**-predicates will allow us to verify whether a vertex is contained in some bag and whether any vertex set in the graph constitutes a bag in its tree decomposition. Each bag will be associated with either a vertex or an edge in the underlying graph (its *witness*) together with some *type*, whose definition depends on the graph class under consideration. The **Parent**-predicate allows for identifying edges in the tree decomposition, i.e. for any two vertex sets S_p and S_c, this predicate will be true if and only if both S_p and S_c are bags in the tree decomposition and S_p is the bag corresponding to the parent node of S_c.

▶ **Definition 8** (MSOL-definable tree decomposition)**.** A tree decomposition $(T = (N, F), X)$ of a graph $G = (V, E)$ is called *existentially MSOL-definable*, if the following are existentially MSOL-definable (with parameters $x_1, \ldots, x_p$ for some constant p).

- **(i)** Each bag $X_p, p \in N$ in the tree decomposition is associated with either a vertex $v \in V$ or an edge $e \in E$ (called its *witness*) and can be identified by one of the following predicates (where $S \subseteq V$ and s and t are constants).
  - **(a)** $\mathbf{Bag}_{\tau_1}(v, S), \ldots, \mathbf{Bag}_{\tau_t}(v, S)$: The vertex set S forms a bag in the tree decomposition of G, i.e. $S = X_p$ for some $p \in N$, it is of type τ_i $(1 \leq i \leq t)$ and its witness is v.
  - **(b)** $\mathbf{Bag}_{\sigma_1}(e, S), \ldots, \mathbf{Bag}_{\sigma_s}(e, S)$: The vertex set S forms a bag in the tree decomposition of G, i.e. $S = X_p$ for some $p \in N$, it is of type σ_i $(1 \leq i \leq s)$ and its witness is e.
- **(ii)** Each edge in F can be identified with a predicate $\mathbf{Parent}(S_p, S_c)$, where $S_p, S_c \subseteq V$: The vertex sets S_p and S_c form bags in (T, X), i.e. $S_p = X_p$ and $S_c = X_c$ for some $p, c \in N$, and p is the parent node of c in T.

▶ **Lemma 9.** *Let (T, X) be an existentially MSOL-definable tree decomposition with parameters $x_1, \ldots, x_p$. There exists a predicate ϕ with zero parameters and p arguments, which is true if and only if the predicates $\boldsymbol{Bag}_{\tau_1}, \ldots, \boldsymbol{Bag}_{\tau_t}$, $\boldsymbol{Bag}_{\sigma_1}, \ldots, \boldsymbol{Bag}_{\sigma_s}$ and **Parent** describe a width-k rooted tree decomposition of an evaluation graph G.*

Proof. The proof can be done analogously to the proof of Lemma 4.7 in [16]. ◀

A fundamental result about definable graph properties, which we use extensively throughout our proofs, states that one can define any edge orientation of partial k-trees in MSOL. For an in-depth study of MSOL-definable edge orientations on graphs, see [9].

▶ **Lemma 10** (Lemma 4.8 in [16])**.** *Any direction over a subset of the edges of an undirected graph of treewidth at most k is existentially MSOL-definable with $k + 2$ parameters.*

Tree Automata for Graphs of Bounded Treewidth

We briefly review the concept of tree automata and recognizability of graph properties for graphs of bounded treewidth. For an introduction to the topic we refer to [13, Chapter 12]. For the formal details of the following notions, the reader is referred to [16].

A tree automaton $\mathcal{A}$ is a finite state machine accepting as an input a tree structure over an alphabet Σ as opposed to words in classical word automata. Formally, $\mathcal{A}$ is a triple $(\mathcal{Q}, \mathcal{Q}_{Acc}, f)$ of a set of states $\mathcal{Q}$, a set of accepting states $\mathcal{Q}_{Acc} \subseteq \mathcal{Q}$ and a transition function f, deriving the state of a node in the input tree $\mathcal{T}$ from the states of its children and its own symbol $s \in \Sigma$. $\mathcal{T}$ is *accepted* by $\mathcal{A}$, if the state of the root node of $\mathcal{T}$ is an element of the accepting states $\mathcal{Q}_{Acc}$.

To recognize a graph property on graphs of treewidth at most k, one encodes a rooted width-k tree decompositions as a labeled tree over a special type of alphabet, in the following denoted by Σ_k (see Definition 3.5, Proposition 3.6 in [16]). We say that a tree automaton over such an alphabet *processes* width-k tree decompositions.

▶ **Definition 11** (Recognizable Graph Properties)**.** Let P denote a graph property. We call P *recognizable* (for graphs of treewidth at most k), if there exists a tree automaton $\mathcal{A}_P$ processing width-k tree decompositions, such that following are equivalent.

(i) (T, X) is a width-k tree decomposition of a graph G with $P(G)$.

(ii) $\mathcal{A}_P$ accepts (the labeled tree over Σ_k corresponding to) (T, X).

Kaller has shown that Courcelle's Conjecture follows immediately from the construction of an MSOL-definable tree decomposition.

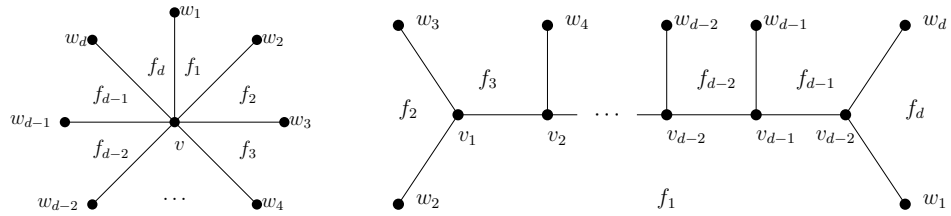
▶ **Lemma 12** (Lemma 5.4 in [16])**.** *Let P denote a graph property, which is recognizable for graphs of bounded treewidth. Suppose that there is an MSOL-definable tree decomposition of width at most k for any partial k-tree G. Then, one can write a CMSOL-sentence Φ, such that $G \models \Phi$ if and only if $P(G)$.*

3 The Main Result

Bodlaender has shown that every k-outerplanar graph has treewidth at most $3k - 1$ [2, Theorem 83], using the following properties of maximal spanning forests of a graph.

▶ **Definition 13** (Vertex and Edge Remember Number)**.** Let $G = (V, E)$ be a graph with maximal spanning forest $T = (V, F)$. The *vertex remember number* of G (with respect to T), denoted by $vr(G, T)$, is the maximum number over all vertices $v \in V$ of fundamental cycles that use v. Analogously, we define the *edge remember number*, denoted by $er(G, T)$.

In particular, Bodlaender gave a constructive proof that the treewidth of a graph is bounded by at most $\max\{vr(G, T), er(G, T) + 1\}$ [2, Theorem 71]. The idea of the proof is to create a bag for each vertex and edge in the spanning tree, containing the vertex itself (or the two endpoints of the edge, respectively) and one endpoint of each edge, whose fundamental cycle uses the corresponding vertex/edge. The tree structure of the decomposition is inherited by the structure of the spanning tree. He then showed, that in a k-outerplanar graph G one can split the vertices of degree $d > 3$ into a path of $d - 2$ vertices of degree three without increasing the outerplanarity index of G (the so-called *vertex expansion step*). In this expanded graph G' one can find a spanning tree of vertex remember number at most $3k - 1$ and edge remember number at most $2k$ [2, Lemmas 81 and 82]. Using [2, Theorem 71], this yields a tree decomposition of width at most $3k - 1$ for G' and by simple replacements



(a) before expansion **(b)** after expansion

Figure 1 Expanding a vertex v, where f_1 is a layer with lowest layer number.

one finds a tree decomposition for G of the same width. A constructive proof of finding such a spanning tree was given by Katsikarelis [17].

The major challenge of defining such a tree decomposition in MSOL lies in the vertex expansion step, since one cannot use artificially created vertices and edges as variables in MSOL-predicates. We give an implicit representation of this step in Section 3.1. We show how to construct an existentially MSOL-definable tree decomposition of a 3-connected k-outerplanar graph in Section 3.2 and for the general case of k-outerplanar graphs in Section 3.3.

3.1 An Implicit Representation of the Vertex Expansion Step

We first define the partition of the vertex set of a k-outerplanar graph into the layers resulting from repeatedly removing the vertices on the outer face.

▶ **Definition 14** (Stripping Layer of a k-Outerplanar Graph)**.** Let G be a k-outerplanar graph. Removing the vertices on the outer face of an embedding of G is called the *stripping step*. When applied repeatedly, the set of vertices being removed in the i-th stripping step is called the i-th *stripping layer* of G, where $1 \leq i \leq k$.

▶ **Lemma 15.** *Let $G = (V, E)$ be a k-outerplanar graph. The partition of V into the stripping layers of G is existentially MSOL-definable with k parameters.*

To avoid increasing the outerplanarity index of a graph during the expansion of a vertex, we need the notion of *layer numbers*.

▶ **Definition 16** (Layer Number)**.** Let $G = (V, E)$ be a planar graph. The *layer number* of a face is defined in the following way. The outer face gets layer number 0. Then, for each other face, we let the layer number be one higher than the minimum layer number of all its adjacent faces.[2]

The expansion step does not preserve facial adjacency, so in order to not increase the outerplanarity index of the graph, one makes sure that all faces are adjacent to a face with lowest layer number. We illustrate the expansion step of a vertex in Figure 1.

Following the ideas of the proofs given in [2, Section 13], we define another type of remember number.

▶ **Definition 17** (Face Remember Number, Face Remember Set)**.** Let $G = (V, E)$ be a planar graph with a given embedding $\mathcal{E}$ and $T = (V, F)$ a maximal spanning forest of G. The *face*

[2] Unless stated otherwise, we call two faces adjacent, if they share an incident vertex.

remember number of G w.r.t. T, denoted by $fr(G,T)$ is the maximum number of fundamental cycles C of G given T, such that $bd_E(f) \cap E(C) \neq \emptyset$, where $bd_E(f)$ denotes the boundary edges of a face f, over all faces f in $\mathcal{E}$, except the outer face. Given a face $f \in \mathcal{E}$, we call the set of edges, whose fundamental cycles intersect the boundary of f the *face remember set*.

In the following, we denote by G a k-outerplanar graph (before expansion) and by G' the graph obtained by expanding vertices of degree $d > 3$. Consider the vertex v_1 in Figure 1b and let e be an edge whose fundamental cycle C_e uses v_1 in some spanning tree of G'. We observe that C_e intersects with one of the face boundaries of f_1, f_2 or f_3. Since v_1 is a vertex in the expanded graph, we know that in each tree decomposition based on a spanning tree of G' there will be a bag containing one endpoint of each edge, whose fundamental cycle intersects with the face boundary of f_1, f_2 or f_3. Using this observation, we can also show that one can find a tree decomposition of a planar graph, whose width is bounded by the edge and face remember number of one of its spanning trees.

▶ **Lemma 18.** *Let $G = (V,E)$ be a planar graph with spanning tree $T = (V,F)$. The treewidth of G is at most* $\max\{er(G,T)+1, 3 \cdot fr(G,T)\}$.

▶ **Lemma 19.** *Let $G = (V,E)$ be a k-outerplanar graph. There exists a spanning tree $T = (V,F)$ of G with $er(G,T) \leq 2k$ and $fr(G,T) \leq k$.*

Proof. The proof can be done analogously to the proof of Lemma 81 in [2]. ◀

3.2 3-Connected k-Outerplanar Graphs

In the previous section we showed that one can construct a tree decomposition of a k-outerplanar graph G, whose width is bounded by the edge and face remember number of G. We will now use these results to show that the construction of such a tree decomposition is existentially MSOL-definable, if we restrict ourselves to 3-connected k-outerplanar graphs.

A classic result by Whitney states that every 3-connected planar graph has a unique embedding [21] (up to the choice of the outer face). Reconstructing this proof, Diestel has shown that the face boundaries of this embedding can be characterized in strictly combinatorial terms.

▶ **Proposition 20** (Proposition 4.2.7 in [12])**.** *The face boundaries of a 3-connected planar graph are precisely its non-separating induced cycles.*[3]

We immediately find the following.

▶ **Proposition 21.** *The face boundaries of a 3-connected planar graph are MSOL-definable.*

Using these observations, we can define predicates encoding an ordering on the incident edges of each vertex.

▶ **Lemma 22.** *Let $G = (V,E)$ be a 3-connected k-outerplanar graph, $v \in V$ with* $\deg(v) > 3$ *and $e_{\mathcal{A}} \in E$ an incident edge, called its* anchor*. There exists an ordering $nb_<(e,f)$, which mimics a clockwise (or counter-clockwise) traversal (in the unique embedding of G) on all incident edges of v, starting at $e_{\mathcal{A}}$, which is existentially MSOL-definable with two parameters $e_{\mathcal{A}}$ and $e'_{\mathcal{A}} \in E$.*

[3] Let $G = (V,E)$ be a connected graph. A vertex set $W \subseteq V$ is called *non-separating induced cycle*, if $G[W]$ is a cycle and $G[V \setminus W]$ is connected.

Note that one can lead an alternative proof of Lemma 22, using the notion of *rotation systems*, introduced in [11]. Furthermore one can see that the relation $\mathrm{nb}_<(e, f)$ is existentially MSOL-definable for a graph G (and not just a single vertex) by replacing the parameters in the formulation of Lemma 22 with the corresponding edge set equivalents.

The construction of the tree decomposition in the proof of the following lemma can be summarized as follows. Given a k-outerplanar graph $G = (V, E)$, one guesses a directed spanning tree $T = (V, F)$ and then constructs a part of the tree decomposition for each vertex $v \in V$ in the following manner. For each incident edge $e \in E$ of v we create one bag which contains one vertex for each edge, whose fundamental cycle uses e and one endpoint of each edge in the face remember set of one fixed incident face of v with minimum layer number. Additionally, for each edge $f \in F$ in the spanning tree of G one creates a bag which contains one endpoint of each edge whose fundamental cycle uses f. We make two bags adjacent, if they either are created due to the same edge in G or if their corresponding edges $e, f \in E$ are direct neighbors in the ordering $\mathrm{nb}_<(e, f)$. For details of the proof the reader is referred to the full text of the paper [14].

▶ **Lemma 23.** *Let $G = (V, E)$ be a 3-connected k-outerplanar graph. G admits an existentially MSOL-definable tree decomposition of width at most $3k$ and maximum degree 3 with $4k + 3$ parameters.*

3.3 Implications of Hierarchical Graph Decompositions to Courcelle's Conjecture

A *block decomposition* of a connected graph G is a tree decomposition, whose bags contain either the endpoints of a single edge or the vertex set of a maximal 2-connected subgraph[4] of G (called the *blocks* of G) or a cut-vertex of G (called the *cuts*) by making a block-bag adjacent to a cut-bag $\{v\}$ if the block bag contains v (see e.g. Section 2.1 in [12]).

Analogously, Tutte showed that given a 2-connected graph (or a block of a connected graph) one can find a *3-block decomposition* into its *2-cuts* and *3-blocks*, the latter of which are vertex sets of either 3-connected graphs or cycles (but not necessarily subgraphs of G, see below), which can be joined in a tree structure in the same way [19, Chapter 11] [20, Section IV.3]. Courcelle showed that both of these decompositions of a graph are MSOL-definable [10] and also proved that one can find an MSOL-definable tree decomposition of width 2, if all 3-blocks of a graph are cycles [10, Corollary 4.11]. In this section, we will use these methods to prove Courcelle's Conjecture for k-outerplanar graphs by showing that the results of the previous section can be applied to define tree decompositions of 3-connected 3-blocks of a k-outerplanar graph.

We will refer to a block decomposition of a graph G, whose 2-connected blocks are replaced by their 3-block decompositions as the *hierarchical decomposition* of G (cf. [10], for an example see Figure 2). For details of how the bags in the resulting decomposition are connected, the reader is referred to the full text of the paper [14].

▶ **Definition 24** (3-Block)**.** Let $G = (V, E)$ be a 2-connected graph, $\mathcal{S}$ a set of 2-cuts of G and $W \subseteq V$. A graph $H = (W, F)$ is called a *3-block*, if it can be obtained by taking the induced subgraph of W in G and for each incident 2-cut $S = \{x, y\} \in \mathcal{S}$, adding the edge $\{x, y\}$ to F (if it is not already present), plus one of the following holds.

[4] Let $G = (V, E)$ be a graph and $W \subseteq V$. $H = G[W]$ is called a *maximal 2-connected subgraph* of G, if $G[W]$ is 2-connected and for all $W' \supset W$, $G[W']$ is not 2-connected.

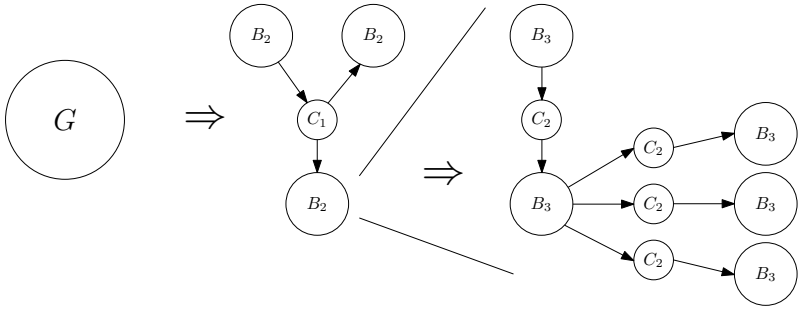


Figure 2 An example hierarchical decomposition of a graph G. A bag labeled C_1 contains a cut-vertex of G, C_2 a 2-cut of G. Bags labeled B_2 contain a 2-block (a single edge or a maximal 2-connected subgraph). If a 2-block contains a maximal 2-connected subgraph of G, it is decomposed further into its 2-cuts and 3-blocks, labeled B_3, which contain either a cycle or a 3-connected 3-block.

(i) H is a cycle of at least three vertices.
(ii) H is a 3-connected graph (referred to as a *3-connected 3-block*).

▶ **Definition 25** (Tutte Decomposition)**.** Let $G = (V, E)$ be a 2-connected graph. A tree decomposition $(T = (N, F), X)$ is called a *Tutte decomposition* of G, if the following hold.

(i) For each $t \in N$, X_t is either a 2-cut (called the *cut bags*) or the vertex set of a 3-block (called the *block bags*).
(ii) Each edge $f \in F$ is incident to precisely one cut bag.
(iii) Each cut bag is adjacent to precisely two block bags.
(iv) Let $t \in N$ denote a cut node with vertex set X_t. Then, t is adjacent to each block node t' with $X_t \subset X_{t'}$.

Tutte has shown that additional restrictions can be formulated on the choice of the set of 2-cuts, such that the resulting decomposition is unique for each graph (for details see the above mentioned literature). In the following, when we refer to *the* Tutte decomposition of a graph, we always mean the one that is unique in this sense, which is also the one that Courcelle defined in his work [10].

▶ **Lemma 26.** *Let $G = (V, E)$ be a 2-connected graph with Tutte decomposition (T, X). G is k-outerplanar, if and only if all 3-connected 3-blocks of (T, X) are at most k-outerplanar.*

The proof of the previous lemma gives the following consequences.

▶ **Corollary 27.** *Let G be a 2-connected graph with Tutte decomposition (T, X).*

(i) *If G is a partial k-tree, then the 3-connected 3-blocks of (T, X) are partial k-trees.*
(ii) *If G is planar, then the 3-connected 3-blocks of (T, X) are planar.*
(iii) *If G is $\mathcal{H}$-minor free, then the 3-connected 3-blocks of (T, X) are $\mathcal{H}$-minor free, where $\mathcal{H}$ is a set of fixed graphs.*

Replacing Edge Quantification by Vertex Quantification

As discussed above, a 3-block is in general not a subgraph of a graph G, as we add edges between the 2-cuts of the Tutte decomposition to turn the 3-blocks into cycles or 3-connected graphs. Since these absent edges cannot be used as variables in MSOL-predicates (which would make our logic non-monadic), we need to find another way to quantify over them.

In [8], Courcelle discusses several structures over which one can define monadic second order logic of graphs, which we will now review.

▶ **Definition 28** (cf. Definition 1.7 in [8]). Let $G = (V, E)$ be a graph. We associate with G two relational structures, denoted by $|G|_1 = \langle V, \text{edg} \rangle$ and $|G|_2 = \langle V \cup E, \text{edg}' \rangle$.

(i) (C)MSOL-predicates over $|G|_1$ only use vertices or vertex sets as variables and we have that $\text{edg}(x, y)$ is true for $x, y \in V$, if and only if there is some edge $e = \{x, y\} \in E$. (C)MSOL-predicates over $|G|_2$ use both vertices and edges and vertex and edge sets as variables. Furthermore, $\text{edg}'(e, x, y)$ is true if and only if $e = \{x, y\}$ and $e \in E$.

(ii) If we can express a graph property in the structure $|G|_1$, we call it *1-definable* and if we can express a graph property in the structure $|G|_2$, we call it *2-definable*.

Clearly, the monadic second order logic we are using throughout this paper is the one represented by the structure $|G|_2$. We use both vertex and edge quantification and one rewrites $\text{Inc}(v, e)$ to $\exists w\ \text{edg}'(e, v, w)$. Since every 1-definable property is trivially also 2-definable, we can conclude that both 1-definability and 2-definability imply (C)MSOL-definability in our sense. Some of the main results of [8] can be summarized as follows.

▶ **Theorem 29** ([8]). *1-Definability equals 2-definability for*

(i) *partial k-trees.*

(ii) *planar graphs.*

(iii) *$\mathcal{H}$-minor free graphs, where $\mathcal{H}$ is a set of fixed graphs.*

Hence, by Theorem 29 we know that we can rewrite each formula using vertex and edge quantification to one only using vertex quantification, if a graph is a member of one of the stated graph classes. We will now show that this result can be used to implicitly quantify over virtual edges of a graph, if these virtual edges can be expressed by an (existentially) MSOL-definable relation. (For a similar application of this result, see [10, Problem 4.10].)

▶ **Lemma 30.** *Let $G = (V, E)$ be a graph which is a member of a graph class $\mathcal{C}$ as stated in Theorem 29 and let P denote a graph property, which is 2-definable by a predicate ϕ_P. Let $E' \subseteq V \times V$ denote a set of virtual edges, such that there is a predicate $edg_{Virt}(v, w)$, which is true if and only if $\{v, w\}$ is a member of E'. Then, P is also 1-definable for the graph $G' = (V, E \cup E')$, if G' is a member of $\mathcal{C}$.*

For the specific case of k-outerplanar graphs, we can now derive the following.

▶ **Corollary 31.** *Let $G = (V, E)$ be a k-outerplanar graph and P a graph property, which is (C)MSOL-definable for 3-connected k-outerplanar graphs. Let B_3 denote a 3-block of G, including the virtual edges between all incident 2-cuts of B_3. Then, P is (C)MSOL-definable for B_3.*

Note that the statements of Lemma 30 and Corollary 31 also hold for existential definability.

Defining the Tree Decomposition of a k-Outerplanar Graph

By Corollary 31 we now know that every graph property, which can be defined for a 3-connected k-outerplanar graph, can also be defined for a 3-block of any k-outerplanar graph G (including its virtual edges). However, there are two more steps we have to take to conclude the proof of our main result, which we will discuss in the current section. First, we have to show that if we are given predicates which encode bounded-width tree decompositions for the 3-connected 3-blocks of a k-outerplanar graph G, then we can define predicates which encode a bounded-width tree decomposition of G.

▶ **Lemma 32.** *Let $G = (V, E)$ be a k-outerplanar graph. G admits an existentially MSOL-definable tree decomposition of width at most $3k + 3$ with a constant number of parameters, if there exist predicates existentially defining width-$3k$ tree decompositions for the 3-connected 3-blocks of G with a constant number of parameters.*

The second obstacle in applying Lemma 23 to define a tree decomposition for G using its (definable) hierarchical graph decomposition is that the number of parameters of all existentially defined predicates has to stay constant. When defining a tree decomposition for a 3-connected k-outerplanar graph in MSOL, one first guesses a rooted spanning tree of G. To avoid guessing a non-constant number of spanning trees, we will find a set of edges $\mathcal{S}_E$, which contains (the edges of) a spanning tree with bounded edge and face remember number for each 3-connected 3-block of G. Furthermore we guess one set $\mathcal{R}_V$, containing one unique vertex for each 3-connected 3-block of G, which we will use as the root of its spanning tree.

▶ **Lemma 33.** *Let $G = (V, E)$ be a planar graph and $G = (V, E \cup E')$ the graph obtained by adding the virtual edges E' of the Tutte decompositions of the 2-connected blocks of G to G. Let $T = (V, F)$ be a spanning tree of G with $er(G, T) \leq \lambda$ and $fr(G, T) \leq \mu$. Let $B_3 = (V_{B_3}, E_{B_3})$ be a 3-connected 3-block of G' (including virtual edges) and $T_{B_3} = T[V_{B_3}]$. One can construct from T_{B_3} a spanning tree $T^*_{B_3}$ of B_3 with $er(B_3, T^*_{B_3}) \leq \lambda$ and $fr(B_3, T^*_{B_3}) \leq \mu$ by adding edges from $E \cup E'$ to T_{B_3}.*

We now give an outline of how to complete the proof of our main result. For the technical details, the reader is referred to the full text [14].

We first show that the statement of Lemma 33 also holds for *directed* spanning trees. It is then trivial to derive the set $\mathcal{R}_V$ of roots of the spanning trees of each 3-connected 3-block. Then we prove that both sets are MSOL-definable with a constant number of parameters. Combining these observations with Lemma 32 we can then conclude that k-outerplanar graphs admit existentially definable tree decompositions of width at most $3k + 3$.

Now we can apply Lemmas 9 and 12 to k-outerplanar graphs and in the light of Courcelle's Theorem [6] we then have our main result.

▶ **Theorem 34.** *Definability equals recognizability for k-outerplanar graphs.*

4 Conclusion

In this paper we have shown that recognizability implies definability in counting monadic second order logic for k-outerplanar graphs, resolving a special case of a conjecture by Courcelle [6]. Starting at the more restrictive case of 3-connected k-outerplanar graphs, we proved that one can use hierarchical graph decompositions to define tree decompositions for general k-outerplanar graphs in monadic second order logic. We have also given indications that this technique might be applicable for other graph classes as well (see Corollary 27), depending on how their tree decompositions are defined in MSOL. 3-Connected graphs often have favorable properties when it comes to defining graph properties in MSOL. For example, in our proof we used the fact that the face boundaries of a 3-connected can be expressed in strictly combinatorial terms and are definable in a straightforward way (see Propositions 20 and 21). Hence, we believe that the techniques presented in this paper can be helpful in resolving the conjecture in its general statement.

Acknowledgements. The second author thanks Bruno Courcelle, Mike Fellows, Pinar Heggernes and Jan Arne Telle for inspiring discussions.

References

1 Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991.

2 Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1-2):1–45, 1998.

3 Hans L. Bodlaender, Pinar Heggernes, and Jan Arne Telle. Recognizability equals definability for graphs of bounded treewidth and bounded chordality. In *Proceedings EUROCOMB 2015*, Electronic Notes in Discrete Mathematics. Elsevier, 2015.

4 Richard B. Borie, R. Gary Parker, and Craig A. Tovey. Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica*, 7(1-6):555–581, 1992.

5 J. Richard Büchi. Weak second-order arithmetic and finite automata. *Mathematical Logic Quarterly*, 6(1-6):66–92, 1960.

6 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990.

7 Bruno Courcelle. The monadic second-order logic of graphs V: On closing the gap between definability and recognizability. *Theoretical Computer Science*, 80(2):153–202, 1991.

8 Bruno Courcelle. The monadic second order logic of graphs VI: On several representations of graphs by relational structures. *Discrete Applied Mathematics*, 54(2-3):117–149, 1994.

9 Bruno Courcelle. The monadic second-order logic of graphs VIII: Orientations. *Annals of Pure and Applied Logic*, 72(2):103–143, 1995.

10 Bruno Courcelle. The monadic second-order logic of graphs XI: Hierarchical decompositions of connected graphs. *Theoretical Computer Science*, 224(1-2):38–53, 1999.

11 Bruno Courcelle. The monadic second-order logic of graphs XII: Planar graphs and planar maps. *Theoretical Computer Science*, 237(1-2):1–32, 2000.

12 Reinhard Diestel. *Graph Theory*. Number 173 in Graduate Texts in Mathematics. Springer, 4th edition, 2012. Corrected reprint.

13 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

14 Lars Jaffke and Hans L. Bodlaender. Definability equals recognizability for k-outerplanar graphs. *ArXiv e-prints*, 2015. http://arxiv.org/abs/1509.08315.

15 Valentine Kabanets. Recognizability equals definability for partial k-paths. In *Proceedings ICALP 1997*, volume 1256 of *LNCS*, pages 805–815. Springer, 1997.

16 Damon Kaller. Definability equals recognizability of partial 3-trees and k-connected partial k-trees. *Algorithmica*, 27(3-4):348–381, 2000.

17 Ioannis Katsikarelis. Computing bounded-width tree and branch decompositions of k-outerplanar graphs. *ArXiv e-prints*, 2013. http://arxiv.org/abs/1301.5896.

18 Denis Lapoire. Recognizability equals monadic second-order definability for sets of graphs of bounded tree-width. In *Proceedings STACS 1998*, volume 1373 of *LNCS*, pages 618–628. Springer, 1998.

19 William T. Tutte. *Connectivity in Graphs*. University of Toronto Press, 1966.

20 William T. Tutte. *Graph Theory*, volume 21 of *Encyclopedia of Mathematics and its Applications*. Addison-Wesley, 1984.

21 Hassler Whitney. Congruent graphs and the connectivity of graphs. *American Journal of Mathematics*, 54:150–168, 1932.

Practical Algorithms for Linear Boolean-width*

Chiel B. ten Brinke, Frank J. P. van Houten, and Hans L. Bodlaender

Department of Computer Science, Utrecht University
PO Box 80.089, 3508 TB Utrecht, The Netherlands
CtenBrinke@gmail.com, Frankv@nhouten.com, H.L.Bodlaender@uu.nl

Abstract

In this paper, we give a number of new exact algorithms and heuristics to compute linear boolean decompositions, and experimentally evaluate these algorithms. The experimental evaluation shows that significant improvements can be made with respect to running time without increasing the width of the generated decompositions. We also evaluated dynamic programming algorithms on linear boolean decompositions for several vertex subset problems. This evaluation shows that such algorithms are often much faster (up to several orders of magnitude) compared to theoretical worst case bounds.

1998 ACM Subject Classification G.2.2 [Discrete Mathematics] Graph Theory – Graph algorithms, F.2.2 [Analysis of Algorithms and Problem Complexity] Nonnumerical Algorithms and Problems – Computations on discrete structures

Keywords and phrases graph decomposition, boolean-width, heuristics, exact algorithms, vertex subset problems

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.187

1 Introduction

Boolean-width is a recently introduced graph parameter [2]. Similarly to treewidth and other parameters, it measures some structural complexity of a graph. Many NP-hard problems on graphs become easy if some graph parameter is small. We need a derived structure which captures the necessary information of a graph in order to exploit such a small parameter. In the case of boolean-width, this is a binary partition tree, referred to as the decomposition tree. However, computing an optimal decomposition tree is usually a hard problem in itself. A common approach to bypass this problem is to use heuristics to compute decompositions with a low boolean-width.

Algorithms for computing boolean decompositions have been studied before in [16, 8, 10, 5], but in this paper we study the specific case of linear boolean decompositions, which are considered in [1, 8, 10]. Linear decompositions are easier to compute and the theoretical running time of algorithms for solving practical problems is lower on linear decompositions than on tree shaped ones. For instance, vertex subset problems can be solved in $O^*(nec^3)$ due to a dynamic programming algorithm by Bui-Xuan et al. [3], but this can be improved to $O^*(nec^2)$ for linear decompositions. Here, nec is the number of d-neighborhood equivalence classes, i.e., the maximum size of the dynamic programming table.

* The research of the third author was partially funded by the Networks programme, funded by the Dutch Ministry of Education, Culture and Science through the Netherlands Organisation for Scientific Research.

10th International Symposium on Parameterized and Exact Computation (IPEC 2015).
Editors: Thore Husfeldt and Iyad Kanj; pp. 187–198

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

We first give an exact algorithm for computing optimal linear boolean decompositions, improving upon existing algorithms, and subsequently investigate several new heuristics through experiments, improving upon the work by Sharmin [10, Chapter 8]. We then study the practical relevance of these algorithms experimentally by solving an instance of a vertex subset problem, investigating the number of equivalence classes compared to the theoretical worst case bounds. Omitted proofs can be found in the full version of this paper [12].

2 Preliminaries

A *graph* $G = (V, E)$ of size n is a pair consisting of a set of n *vertices* V and a set of *edges* E. The *neighborhood* of a vertex $v \in V$ is denoted by $N(v)$. For a subset $A \subseteq V$ we denote the neighborhood by $N(A) = \bigcup_{v \in A} N(v)$. In this paper we only consider simple, undirected graphs and assume we are given a total ordering on the vertices of a graph G. For a subset $A \subseteq V$ we denote the *complement* by $\overline{A} = V \setminus A$. A partition $(A, \overline{A})$ of V is called a *cut* of the graph. Each cut $(A, \overline{A})$ of G induces a bipartite subgraph $G[A, \overline{A}]$. The *neighborhood across a cut* $(A, \overline{A})$ for a subset $X \subseteq A$ is defined as $N(X) \cap \overline{A}$.

▶ **Definition 1** (Unions of neighborhoods)**.** Let $G = (V, E)$ be a graph and $A \subseteq V$. We define the set of *unions of neighborhoods across a cut* $(A, \overline{A})$ as

$$\mathcal{UN}(A) = \left\{ N(X) \cap \overline{A} \,\middle|\, X \subseteq A \right\}.$$

The number of unions of neighborhoods is symmetric for a cut $(A, \overline{A})$, i.e., $|\mathcal{UN}(A)| = |\mathcal{UN}(\overline{A})|$ [6, Theorem 1.2.3]. Furthermore, for any cut $(A, \overline{A})$ of a graph G it holds that $|\mathcal{UN}(A)| = \#\mathcal{MIS}(G[A, \overline{A}])$, where $\#\mathcal{MIS}(G)$ is the number of maximal independent sets in G [16, Theorem 3.5.5].

▶ **Definition 2** (Decomposition tree)**.** A *decomposition tree* of a graph $G = (V, E)$ is a pair (T, δ), where T is a full binary tree and δ is a bijection between the leaves of T and vertices of V. If a is a node and L are its leaves, we write $\delta(a) = \bigcup_{l \in L} \delta(l)$. So, for the root node r of T it holds that $\delta(r) = V$. Furthermore, if nodes a and b are children of a node w, then $(\delta(a), \delta(b))$ is a partition of $\delta(w)$.

In this paper we consider a special type of decompositions, namely *linear decompositions*.

▶ **Definition 3** (Linear decomposition)**.** A *linear decomposition*, or *caterpillar decomposition*, is a decomposition tree (T, δ) where T is a full binary tree and for which each internal node of T has at least one leaf as a child. We can define such a linear decomposition through a linear ordering $\pi = \pi_1, \ldots, \pi_n$ of the vertices of G by letting δ map the i-th leaf of T to π_i.

▶ **Definition 4** (Boolean-width)**.** Let $G = (V, E)$ be a graph and $A \subseteq V$. The *boolean dimension of* A is a function bool-dim : $2^V \to \mathbb{R}$.

$$\text{bool-dim}(A) = \log_2 |\mathcal{UN}(A)|.$$

Let (T, δ) be a decomposition of a graph G. We define the *boolean-width* of (T, δ) as the maximum boolean dimension over all cuts induced by nodes of (T, δ).

$$\text{boolw}(T, \delta) = \max_{w \in T} \text{bool-dim}(\delta(w))$$

The boolean-width of G is defined as the minimum boolean-width over all possible full decompositions of G, while the *linear boolean-width* of a graph $G = (V(G), E(G))$ of size n is defined as the the minimum boolean-width over all linear decompositions of G.

$$\text{boolw}(G) = \min_{(T,\delta)\ of\ G} \text{boolw}(T, \delta)$$

$$\text{lboolw}(G) = \min_{\text{linear } (T,\delta) \text{ of } G} \text{boolw}(T, \delta)$$

It is known that for any graph G it holds that $\text{boolw}(G) \leq \textit{treewidth}(G)+1$ [16, Theorem 4.2.8]. The linear variant of treewidth is called *pathwidth* [9], or pw for short.

▶ **Theorem 5.** *For any graph G it holds that* $\text{lboolw}(G) \leq \text{pw}(G) + 1$.

The algorithms in this paper make extensive use of sets and set operations, which can be implemented efficiently by using bitsets. We assume that bitset operations take $O(n)$ time and need $O(n)$ space, even though in practice this may come closer to $O(1)$. If one assumes that these requirements are constant, several time and space bounds in this paper improve by a factor n.

In this paper we assume that the graph G is connected, since if the graph consists of multiple connected components we can simply compute a linear decomposition for each connected component, after which we glue them together, in any arbitrary order.

3 Exact Algorithms

We can characterize the problem of finding an optimal linear decomposition by the following recurrence relation, in which P is a function mapping a subset of vertices A to the linear boolean-width of the induced subgraph $G[A, \overline{A}]$.

$$\begin{aligned} P(\{v\}) = |\mathcal{UN}(\{v\})| &= \begin{cases} 1 & \text{if } N(v) = \emptyset \\ 2 & \text{if } N(v) \neq \emptyset \end{cases} \\ P(A) &= \min_{v \in A}\{\max\{|\mathcal{UN}(A)|, P(A \setminus \{v\})\}\} \end{aligned} \tag{1}$$

The boolean-width of the graph G is now given by $\log_2(P(V))$. Adaptation of existing techniques lead to the following algorithms for linear boolean-width, upon we hereafter improve:

- With dynamic programming a running time of $O(2.7284^n)$ is achieved [12, Theorem 19].
- With adaptation of the exact algorithm for boolean-width by Vatshelle [16], a running time of $O(n^3 \cdot 2^{n+\text{lboolw}(G)})$ is achieved [12, Theorem 20].

3.1 Improving the running time

We present a faster and easier way to precompute for all cuts $A \subseteq V$ the value $|\mathcal{UN}(A)|$, which results in a new algorithm displayed in Algorithm 2. In the following it is important that the $\mathcal{UN}$ sets are implemented as hashmaps, which will only save distinct neighborhoods.

Algorithm 1 Compute $\mathcal{UN}(X \cup \{v\})$ given $\mathcal{UN}(X)$.

```
1: procedure INCREMENT-UN(G, X, UN_X, v)
2:     U ← ∅
3:     for S ∈ UN_X do
4:         U ← U ∪ {S \ {v}}
5:         U ← U ∪ {(S \ {v}) ∪ (N(v) ∩ (X̄ \ {v}))}
6:     return U
```

▶ **Lemma 6.** *The procedure Increment-UN is correct and runs in $O(n \cdot |\mathcal{UN}_X|)$ time using $O(n \cdot |\mathcal{UN}_X|)$ space.*

Algorithm 2 Return $\text{lboolw}(G)$, if it is smaller than $\log K$, otherwise return ∞.

```
1:  procedure Incremental-UN-exact(G, K)
2:      T_UN(∅) ← 0
3:      Compute-count-UN(G, K, T_UN, ∅, {∅})
4:
5:      P(X) ← ∞, for all X ⊆ V
6:      P(∅) ← 0
7:
8:      for i ← 0, ..., |V| − 1 do
9:          for X ⊆ V of size i do
10:             for v ∈ V \ X do
11:                 Y ← X ∪ {v}
12:                 if P(X) ≤ K then
13:                     P(Y) ← min(P(Y), max(T_UN(Y), P(X)))
14:
15:     return log2(P(V))
16:
17: procedure Compute-count-UN(G, K, T_UN, X, UN_X)
18:     for v ∈ V \ X do
19:         Y ← X ∪ {v}
20:         if T_UN(Y) is not defined then
21:             UN_Y ← Increment-UN(G, X, UN_X, v)
22:             T_UN(Y) ← |UN_Y|
23:             if T_UN(Y) ≤ K then
24:                 Compute-count-UN(G, K, T_UN, Y, UN_Y)
```

▶ **Theorem 7.** *Given a graph G, Algorithm 2 can be used to compute* $\text{lboolw}(G)$ *in* $O(n \cdot 2^{n+\text{lboolw}(G)})$ *time using* $O(n \cdot 2^n)$ *space.*

This new algorithm improves upon the previously best time [12, Theorem 20] by a factor n^2, while the space requirements stay the same. Since the tightest known upperbound for linear boolean-width is $\frac{n}{2} - \frac{n}{143} + O(1)$ [8], this algorithm can be slower than dynamic programming, since $O(2^{n+\frac{n}{2}-\frac{n}{143}+O(1)}) = O(2.8148^{n+O(1)}) \supsetneq O(2.7284^n)$, but this is very unlikely to happen in practice.

4 Heuristics

4.1 Generic form of the heuristics

The goal when using a heuristic is to find a linear ordering of the vertices in a graph in such a way that the decomposition that corresponds to this ordering will be of low boolean-width. A basic strategy to accomplish this is to start the ordering with some vertex and then by some selection criteria append a new vertex to the ordering that has not been appended yet. This strategy is used in heuristics introduced by Sharmin [10, Chapter 8].

At any point in the algorithm we denote the set of all vertices contained in the ordering by *Left*, and the remaining vertices by *Right*. While *Right* is not empty, we choose a vertex from a candidate set *Candidates* $\subseteq$ *Right*, based on a set of trivial cases, and, if no trivial case applies, by making a local greedy choice using a score function that indicates the quality of the current state *Left*, *Right*.

The choice of the initial vertex can be of great influence on the quality of the decomposition. Sharmin proposes to use a double breadth first search (BFS) in order to select this vertex, but since we will see in Chapter 5 that applications are a lot more expensive in terms of running time, it is wise to use all possible starting vertices when trying to find a good decomposition.

4.1.1 Pruning

Starting from multiple initial vertices allows us to do some pruning. If we notice during the algorithm that the score of the decomposition that is being constructed exceeds the score of the best decomposition found so far, we can stop immediately and move to the next initial vertex. For this reason, it is wise to start with the most promising initial vertices (e.g. obtained by the double BFS method), and after that try all other initial vertices.

4.1.2 Candidates

The most straightforward choice for the set *Candidates* is to take *Right* entirely. However, we may do unnecessary work here, since vertices that are more than 2 steps away from any vertex in $Left$ cannot decrease the size of $\mathcal{UN}$. This means that they should never be chosen by a greedy score function, which means that we can skip them right away. By this reasoning, the set of *Candidates* can be reduced to $N^2(Left) \cap Right = N(Left \cup N(Left)) \cap Right$. Especially for larger sparse graphs, this can significantly decrease the running time.

4.1.3 Trivial cases

A vertex is chosen to be the next vertex in the ordering if it can be guaranteed that it is an optimal choice by means of a trivial case. Lemma 8 generalizes results by Sharmin [10], since the two trivial cases given by her are subcases of our lemma, namely $X = \emptyset$ and $X = \{u\}$ for all $u \in Left$. Note that we can add a wide range of trivial cases by varying X, such as $X = Left$ and $\forall u, w \in Left : X = \{u, w\}$, but this will increase the complexity of the algorithm.

▶ **Lemma 8.** *Let $X \subseteq Left$. If $\exists v \in Right$ such that $N(v) \cap Right = N(X) \cap Right$, then choosing v will not change the boolean-width of the resulting decomposition.*

4.1.4 Relative Neighborhood Heuristic

For a cut $(Left, Right)$ and a vertex v define

$$Internal(v) = (N(v) \cap N(Left)) \cap Right$$
$$External(v) = (N(v) \setminus N(Left)) \cap Right$$

In the original formulation by Sharmin [10] $\frac{|External(v)|}{|Internal(v)|}$ is used as a score function. However, if we use $\frac{|External(v)|}{|Internal(v)|+|External(v)|} = \frac{|External(v)|}{|N(v) \cap Right|}$ we get the same ordering by Lemma 9, without having an edge case for dividing by zero. Furthermore, in contrast to Sharmin's proposal of checking for each vertex $w \in N(v)$ if $w \in N(Left) \cap Right$ or not, we can compute these sets directly by performing set operations. We will refer to this heuristic by RelativeNeighborhood.

▶ **Lemma 9.** *The mapping $\frac{a}{b} \mapsto \frac{a}{a+b}$ is order preserving.*

Two variations on this heuristic can be obtained through the score functions $\frac{|External(v)|}{|N(v)|}$ and $1 - \frac{|Internal(v)|}{|N(v)|}$, which work slightly better for sparse random graphs and extremely well for dense random graphs respectively. We will refer to these two variations by RelativeNeighborhood$_2$ and RelativeNeighborhood$_3$.

One can easily see that the running time of these three algorithms is $O(n^3)$ and the required space amounts to $O(n)$. Notice however that this algorithm only gives us a decomposition. If we need to know the corresponding boolean-width we need to compute it afterwards, for instance by iteratively applying Increment-UN on the vertices in the decomposition, and taking the maximum value. This would require an additional $O(n^2 \cdot 2^k)$ time and $O(n \cdot 2^k)$ space, where k is the boolean-width of the decomposition.

4.1.5 Least Cut Value Heuristic

The LeastCutValue heuristic by Sharmin [10] greedily selects the next vertex $v \in Right$ that will have the smallest boolean dimension across the cut $(Left \cup \{v\}, Right \setminus \{v\})$. This vertex is obtained by constructing the bipartite graph $BG = G[Left \cup \{v\}, Right \setminus \{v\}]$ for each $v \in Right$, and counting the number of maximal independent sets of BG using the CCM_{IS} [7] algorithm on BG, with the time of CCM_{IS} being exponential in n.

4.1.6 Incremental Unions of Neighborhoods Heuristic

Generating a bipartite graph and then calculating the number of maximal independent sets is a computational expensive approach. A different way to compute the boolean dimension of each cut is by reusing the neighborhoods from the previous cut, similarly to Incremental-UN-exact. We present a new algorithm, called the Incremental-UN-heuristic, in Algorithm 3. A useful property of this algorithm is that the running time is output sensitive. It follows that if a decomposition is not found within reasonable time, then the decomposition that would have been generated is not useful for practical algorithms.

▶ **Theorem 10.** *The* Incremental-UN-heuristic *procedure runs in $O(n^3 \cdot 2^k)$ time using $O(n \cdot 2^k)$ space, where k is the boolean-width of the resulting linear decomposition.*

5 Vertex subset problems

Boolean decompositions can be used to efficiently solve a class of vertex subset problems called (σ, ρ) vertex subset problems, which were introduced by Telle [11]. This class of problems consists of finding a (σ, ρ)-set of maximum or minimum cardinality and contains well known problems such as the maximum independent set, the minimum dominating set and the maximum induced matching problem. The running time of the algorithm for solving these problems is $O(n^4 \cdot nec_d(T, \delta)^3)$ [3], where $nec_d(T, \delta)$ is the number of equivalence classes of a problem specific equivalence relation, which can be bounded in terms of boolean-width. In Section 6 we investigate how close the value of $nec_d(T, \delta)$ comes to any of the theoretical bounds.

Algorithm 3 Greedy heuristic that incrementally keeps track of the Unions of Neighborhoods.

```
1:  procedure INCREMENTAL-UN-HEURISTIC(G, init)
2:      Decomposition ← (init)
3:      Left, Right ← {init}, V \ {init}
4:      UN_Left ← {∅, N(init) ∩ Right}
5:      while Right ≠ ∅ do
6:          Candidates ← set returned by candidate set strategy
7:          if there exists v ∈ Candidates belonging to a trivial case then
8:              chosen ← v
9:              UN_chosen ← INCREMENT-UN(G, Left, UN_Left, v)
10:         else
11:             for all v ∈ Candidates do
12:                 UN_v ← INCREMENT-UN(G, Left, UN_Left, v)
13:                 if chosen is undefined or |UN_v| < |UN_chosen| then
14:                     chosen ← v
15:                     UN_chosen ← UN_v
16:         Decomposition ← Decomposition · chosen
17:         Left ← Left ∪ {chosen}
18:         Right ← Right \ {chosen}
19:         UN_Left ← UN_chosen
20:     return Decomposition
```

5.1 Definitions

▶ **Definition 11** ((σ,ρ)-set)**.** Let $G=(V,E)$ be a graph. Let σ and ρ be finite or co-finite subsets of $\mathbb{N}$. A subset $X \subseteq V$ is called a (σ,ρ)-set if the following holds

$$\forall v \in V : |N(v) \cap X| \in \begin{cases} \sigma & \text{if } v \in X, \\ \rho & \text{if } v \in V \setminus X. \end{cases}$$

In order to confirm if a set X is a (σ,ρ)-set we have to count the number of neighbors each vertex $v \in V$ has in X, where it suffices to count up until a certain number of neighbors. As an example, when we want to confirm if a set X is an independent set, which is equivalent to checking if X is a $(\{0\},\mathbb{N})$-set, it is irrelevant if a vertex v has more than one neighbor in X. We capture this property in the function $d : 2^{\mathbb{N}} \to \mathbb{N}$, which is defined as follows:

▶ **Definition 12** (d-function)**.** Let $d(\mathbb{N}) = 0$. For every finite or co-finite set $\mu \subseteq \mathbb{N}$, let $d(\mu) = 1 + \min(\max_{x\in\mathbb{N}} x : x \in \mu, \max_{x\in\mathbb{N}} x : x \notin \mu)$. Let $d(\sigma,\rho) = \max(d(\sigma), d(\rho))$.

▶ **Definition 13** (d-neighborhood)**.** Let $G=(V,E)$ be a graph. Let $A \subseteq V$ and $X \subseteq A$. The *d-neighborhood* of X with respect to A, denoted by $N_A^d(X)$, is a multiset of vertices from $\overline{A}$, where a vertex $v \in \overline{A}$ occurs $\min(d, |N(v) \cap X|)$ times in $N_A^d(X)$. A d-neighborhood can be represented as a vector of length $|\overline{A}|$ over $\{0,1,\ldots,d\}$.

▶ **Definition 14** (d-neighborhood equivalence)**.** Let $G=(V,E)$ be a graph and $A \subseteq V$. Two subsets $X, Y \subseteq A$ are said to be *d-neighborhood equivalent* with respect to A, denoted by $X \equiv_A^d Y$, if it holds that $\forall v \in \overline{A} : \min(d, |X \cap N(v)|) = \min(d, |Y \cap N(v)|)$. The number of equivalence classes of a cut $(A,\overline{A})$ is denoted by $nec(\equiv_A^d)$. The number of equivalence classes $nec_d(T,\delta)$ of a decomposition (T,δ) is defined as $\max(nec(\equiv_A^d), nec(\equiv_{\overline{A}}^d))$ over all cuts $(A,\overline{A})$ of (T,δ).

Note that $N_A^1(X) = N(X) \cap \overline{A}$. It can then be observed that $|\mathcal{UN}(A)| = nec(\equiv_A^1)$ [16, Theorem 3.5.5] Also note that $X \equiv_A^d Y$ if and only if $N_A^d(X) = N_A^d(Y)$.

5.2 Bounds on the number of equivalence classes

We present a brief overview of the most relevant bounds that are currently known, for which we make use of a *twin class partition* of a graph.

▶ **Definition 15** (Twin class partition). Let $G = (V, E)$ be a graph of size n and let $A \subseteq V$. The *twin class partition* of A is a partition of A such that $\forall x, y \in A$, x and y are in the same partition class if and only if $N(x) \cap \overline{A} = N(y) \cap \overline{A}$. The number of partition classes of A is denoted by $ntc(A)$ and it holds that $ntc(A) \leq \min(n, 2^{\text{bool-dim}(A)})$ [2].

For all bounds listed below, let $G = (V, E)$ be a graph of size n and let d be a non-negative integer. Let $(A, \overline{A})$ be a cut induced by any node of a decomposition (T, δ) of G, and let $k = \text{bool-dim}(A) = nec(\equiv_A^1)$.

▶ **Lemma 16** ([3, Lemma 5]). $nec(\equiv_A^d) \leq 2^{d \cdot k^2}$.

▶ **Lemma 17** ([16, Lemma 5.2.2]). $nec(\equiv_A^d) \leq (d+1)^{\min(ntc(A), ntc(\overline{A}))}$.

▶ **Lemma 18.** $nec(\equiv_A^d) \leq ntc(A)^{d \cdot k}$.

By Lemma 16 we conclude that we can solve (σ, ρ) problems in $O^*(8^{dk^2})$. This shows that applications are more computationally expensive than using heuristics to find a decomposition.

6 Experiments

The experiments in this section are performed on a 64-bit Windows 7 computer, with a 3.40 GHz Intel Core i5-4670 CPU and 8GB of RAM. We implemented the algorithms using the C# programming language and compiled our programs using the *csc* compiler that comes with Visual Studio 12.0.[1]

6.1 Comparing Heuristics on random graphs

We will look at the performance of heuristics on randomly generated graphs, for which we used the Erdös-Rényi-model [4] to generate a fixed set of random graphs with varying edge probabilities.

In these experiments we start a heuristic once for each possible initial vertex, so n times in total. For the RelativeNeighborhood heuristic we select the best decomposition based upon the sum of the score function for all cuts, since computing all actual linear boolean-width values would take $O(n^3 \cdot 2^k)$ time, thereby removing the purpose of this polynomial time heuristic. For the set $Candidates$ we take $N^2(Left) \cap Right$, which avoids that we exclude certain optimal solutions, as opposed to Sharmin [10], who restricted this set to $N(Left) \cap Right$. However, this does not affect the results significantly.

We let the edge probability vary between 0.05 and 0.95 with steps of size 0.05. For each edge probability value, we generated 20 random graphs. The result per edge probability is taken to be the average boolean-width over these 20 graphs, which are shown in Figure 1. It

[1] Source code of our implementations can be found on `https://github.com/Chiel92/boolean-width` and `https://github.com/FrankvH/BooleanWidth`

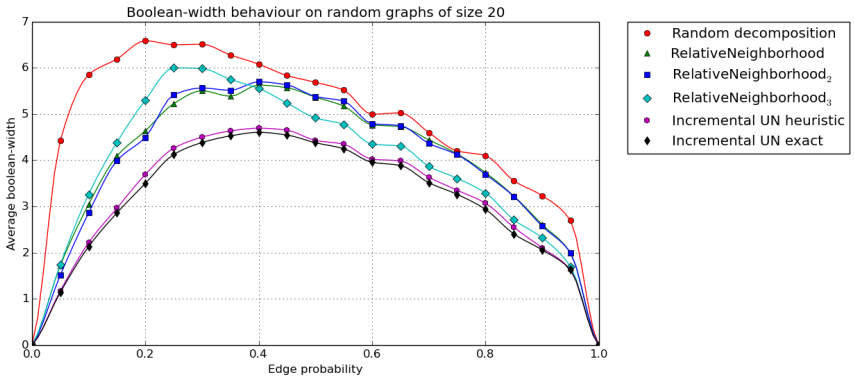


Figure 1 Performance of different heuristics on random generated graphs consisting of 20 vertices, with varying edge probabilities, in terms of linear boolean-width.

can be observed that the INCREMENTAL-UN-HEURISTIC procedure performs near optimal. Furthermore we see that the RELATIVENEIGHBORHOOD variants perform somewhere in between the optimal value and the value of random decompositions.

6.2 Comparing heuristics on real-world graphs

In order to get an idea of how the INCREMENTAL-UN-HEURISTIC compares to existing heuristics we compare them by both the boolean-width of the generated decomposition and the time needed for computation. We cannot compare the heuristics to the optimal solution, because computing an exact decomposition is not feasible on these graphs. The graphs that were used come from *Treewidthlib* [13], a collection of graphs that are used to benchmark algorithms using treewidth and related graph problems.

We ran the three different heuristics mentioned in Section 4 with *Candidates* = *Right* and with an additional two variations on the INCREMENTAL-UN-HEURISTIC (IUN) by varying the set of start vertices. The first variation, named 2-IUN, has two start vertices which are obtained through a single and double BFS respectively. The n-IUN heuristic uses all possible start vertices. For all other heuristics we obtained the start vertex through performing a double BFS. In Table 1 and 2 we present the results of our experiments.

It is expected that the IUN heuristic and LEASTCUTVALUE heuristic give the same linear boolean-width, since both these heuristics greedily select the vertex that minimizes the boolean dimension. The RELATIVENEIGHBORHOOD heuristic performs worse than all other heuristics in nearly all cases. While the difference might not seem very large, note that algorithms parameterized by boolean-width are exponential in the width of a decomposition. The 2-IUN heuristic outperforms IUN in three cases while n-IUN gives a better decomposition in 11 out of 13 cases, which shows that a good initial vertex is of great influence on the width of the decomposition.

Looking at the times displayed in Table 2 for computing each decomposition we see that the RELATIVENEIGHBORHOOD heuristic is significantly faster. This is to be expected because of the $O(n^3)$ time, compared to the exponential time for all other heuristics. The interesting comparison that we can make is the difference between the IUN heuristic and LEASTCUTVALUE heuristic. While both of these heuristics give the same decomposition, IUN is significantly faster. Additionally, even 2-IUN and n-IUN are often faster than the LEASTCUTVALUE heuristic.

Table 1 Linear boolean-width of the decompositions returned by different heuristics.

Graph	$\|V\|$	Edge Density	Relative	LeastCut	IUN	2-IUN	n-IUN
barley	48	0.11	5.70	5.91	5.91	4.70	4.58
pigs-pp	48	0.12	10.35	7.13	7.13	7.13	6.64
david	87	0.11	9.38	6.27	6.27	6.27	5.86
celar04-pp	114	0.08	11.67	7.27	7.27	7.27	7.27
1bkb-pp	127	0.18	16.81	9.98	9.98	9.53	9.53
miles1500	128	0.64	8.17	5.58	5.58	5.58	5.29
celar10-pp	133	0.07	10.32	11.95	11.95	7.64	6.91
munin2-pp	167	0.03	15.17	9.61	9.61	9.61	7.61
mulsol.i.5	186	0.23	7.55	5.29	5.29	5.29	3.58
zeroin.i.2	211	0.16	7.92	4.46	4.46	4.46	3.81
boblo	221	0.01	19.00	4.32	4.32	4.32	4.00
fpsol2.i-pp	233	0.40	5.58	6.07	6.07	5.78	4.81
munin4-wpp	271	0.02	13.04	9.27	9.27	9.27	7.61

Table 2 Time in seconds of the heuristics used to find linear boolean decompositions.

Graph	$\|V\|$	Edge Density	Relative	LeastCut	IUN	2-IUN	n-IUN
barley	48	0.11	< 0.01	0.18	0.01	0.02	0.16
pigs-pp	48	0.12	< 0.01	0.76	0.02	0.04	0.52
david	87	0.11	0.02	3.15	0.04	0.06	1.62
celar04-pp	114	0.08	0.04	5.73	0.14	0.23	9.85
1bkb-pp	127	0.18	0.06	198.05	1.14	4.18	107.32
miles1500	128	0.64	0.06	44.57	0.10	0.14	7.05
celar10-pp	133	0.07	0.06	8.93	1.96	4.72	18.43
munin2-pp	167	0.03	0.11	3.81	0.80	3.37	30.21
mulsol.i.5	186	0.23	0.09	37.88	0.13	0.27	8.80
zeroin.i.2	211	0.16	0.06	18.70	0.09	0.11	5.85
boblo	221	0.01	0.29	3.39	0.28	0.56	46.22
fpsol2.i-pp	233	0.40	0.18	189.11	0.36	0.74	56.63
munin4-wpp	271	0.02	0.61	57.87	1.98	6.66	367.37

6.3 Vertex subset experiments

We have used the linear decompositions given by the n-IUN heuristic to compute the size of the maximum induced matching (MIM) in a selection of graphs, of which the results are presented in Table 3. The maximum induced matching problem is defined as finding the largest $(\{1\}, \mathbb{N})$ set, with $d(\{1\}, \mathbb{N}) = 2$. The choice for the MIM problem is arbitrary, any vertex subset problem with $d = 2$ will have the same number of equivalence classes and therefore they all require the same time when computing a solution. We present the computed value of $nec_d(T, \delta)$, together with theoretical upperbounds. For $d = 2$ a tight upperbound in terms of boolean-width is not known. Note that we take the logarithm of each value, since we find this value easier to interpret and compare to other graph parameters. We let $UB_1 = 2^{d \cdot \text{boolw}^2}$, $UB_2 = (d+1)^{\min ntc}$ and $UB_3 = ntc^{d \cdot \text{boolw}}$, with $ntc = \max_{w \in T} ntc(\delta(w))$ and $\min ntc = \max_{w \in T} \min(ntc(\delta(w)), ntc(\overline{\delta(w)}))$.

The column *MIM* displays the size of the MIM in the graph, while the time column indicates the time needed to compute this set. Missing values for *nec* and MIM are caused

Table 3 Results of using the algorithm by Bui-Xuan et al. [3] for solving (σ, ρ) problems on graphs, using decompositions obtained through the n-IUN heuristic.

Graph	boolw	$\log_2(nec)$	$\log_2(UB_1)$	$\log_2(UB_2)$	$\log_2(UB_3)$	*MIM*	Time (s)
barley	4.58	7.00	42.04	12.68	27.51	22	3
pigs-pp	6.64	10.31	88.28	19.02	49.17	22	1147
david	5.86	9.37	68.63	22.19	44.61	34	919
celar04-pp	7.27	11.15	105.61	28.53	65.74	–	–
1bkb-pp	9.53	–	181.47	52.30	98.49	–	–
miles1500	5.29	9.30	55.87	34.87	49.69	8	4038
celar10-pp	6.91	10.34	95.41	25.36	59.70	50	10179
munin2-pp	7.61	11.82	115.97	19.02	54.60	–	–
mulsol.i.5	3.58	6.11	25.70	14.26	24.80	46	22
zeroin.i.2	3.81	6.58	28.99	20.60	28.18	30	59
boblo	4.00	6.17	32.00	9.51	20.68	148	41
fpsol2.i-pp	4.81	8.07	46.22	22.19	36.61	46	934
munin4-wpp	7.61	12.13	115.97	19.02	57.98	–	–

by a lack of internal memory, because of the $O^*(nec_d(T,\delta)^2)$ space requirement. One can immediately see that there is a large gap between the upperbound for nec_2 in terms of boolean-width and nec_2 itself. Another interesting observation we can make by looking at the graphs zeroin.i.2 and boblo, is that a lower boolean-width does not imply a lower nec_2. We even encountered this for decompositions of the same graph: for the graph barley we observed $\text{boolw}(T,\delta) = 4.58$ and $\text{boolw}(T',\delta') = 4.81$, while $\log_2(nec_2(T,\delta)) = 7.00$ and $\log_2(nec_2(T',\delta')) = 6.75$. This suggests that this upperbound does not justify minimizing nec_2 through boolean-width in practice.

7 Conclusion

We have presented a new heuristic and a new exact algorithm for finding linear boolean decompositions. The heuristic has a running time that is several orders of magnitude lower than the previous best heuristic and finds a decomposition in output sensitive time. This means that if a decomposition is not found within reasonable time, then the decomposition that would have been generated is not useful for practical algorithms. Running the new heuristic once for every possible starting vertex results in significantly better decompositions compared to existing heuristics.

We have seen that if $\text{lboolw}(T,\delta) < \text{lboolw}(T',\delta')$, then there is no guarantee that $nec(T,\delta) < nec(T',\delta')$. While in general it holds that minimizing boolean-width results in a low value of number of equivalence classes, we think that it can be worthwhile to focus on minimizing the nec_d instead of the boolean-width when solving vertex subset problems. However, the number of equivalence classes is not symmetric, i.e., for a cut $(A, \overline{A})$ $nec_d(A) \neq nec_d(\overline{A})$, which makes it harder to develop fast heuristics that focus on minimizing nec_d since we need to keep track of both the equivalence classes of A and $\overline{A}$.

Further research can be done in order to obtain even better heuristics and better upperbounds on both the linear boolean-width and boolean-width on graphs. For instance, combining properties of the Incremental-UN-heuristic and the RelativeNeighborhood heuristic might lead to better decompositions, as they make use of complementary features of a graph. Another approach for obtaining good decompositions could be a branch

and bound algorithm that makes us of trivial cases that are used in the heuristics. To decrease the time needed by the heuristics one can investigate reduction rules for linear boolean-width. While most reduction rules introduced by Sharmin [10] for boolean-width do not hold for linear boolean-width, they can still be used on a graph after which we can use our heuristic on the reduced graph. Although the resulting decomposition after reinserting the reduced vertices will not be linear, the asymptotic running time for applications does not increase [14]. Another topic of research is to compare the performance of vertex subset algorithms parameterized by boolean-width to algorithms parameterized by treewidth [15].

References

1 R. Belmonte and M. Vatshelle. Graph classes with structured neighborhoods and algorithmic applications. *Theoretical Computer Science*, 511:54–65, 2013. Exact and Parameterized Computation.

2 B.-M. Bui-Xuan, J. A. Telle, and M. Vatshelle. Boolean-width of graphs. In *IWPEC 2009*, volume 5917 of *LNCS*, pages 61–74. Springer, 2009.

3 B.-M. Bui-Xuan, J. A. Telle, and M. Vatshelle. Fast dynamic programming for locally checkable vertex subset and vertex partitioning problems. *Theoretical Computer Science*, 511:66–76, 2013.

4 P. Erdös and A. Rényi. On random graphs. *Publicationes Mathematicae 6: 290–297*, 1959.

5 E. M. Hvidevold, S. Sharmin, J. A. Telle, and M. Vatshelle. Finding good decompositions for dynamic programming on dense graphs. In *IWPEC 2012*, volume 7112 of *LNCS*, pages 219–231. Springer, 2012.

6 K. H. Kim. *Boolean Matrix Theory and its Applications*. Marcel Dekker, 1982.

7 F. Manne and S. Sharmin. Efficient counting of maximal independent sets in sparse graphs. In *Experimental Algorithms*, volume 7933 of *LNCS*, pages 103–114. Springer, 2013.

8 Y. Rabinovich, J. A. Telle, and M. Vatshelle. Upper bounds on boolean-width with applications to exact algorithms. In *IWPEC 2013*, volume 8246 of *LNCS*, pages 308–320. Springer, 2013.

9 N. Robertson and P. D. Seymour. Graph minors. I. Excluding a forest. *Journal of Combinatorial Theory, Series B*, 35(1):39–61, 1983.

10 S. Sharmin. *Practical Aspects of the Graph Parameter Boolean-width*. PhD thesis, University of Bergen, Norway, 2014.

11 J. A. Telle. Complexity of domination-type problems in graphs. *Nordic Journal of Computing*, 1(1):157–171, 1994.

12 Ch. B. Ten Brinke, F. J. P. van Houten, and H. L. Bodlaender. Practical Algorithms for Linear Boolean-width. *ArXiv e-prints ArXiV:1509.07687*, 2015.

13 Treewidthlib. http://www.staff.science.uu.nl/∼bodla101/treewidthlib/. A benchmark for algorithms for treewidth and related graph problems.

14 F. J. P. van Houten. Experimental research and algorithmic improvements involving the graph parameter boolean-width. Master's thesis, Utrecht University, The Netherlands, 2015.

15 J. M. M. van Rooij, H. L. Bodlaender, and P. Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In *Algorithms – ESA 2009*, volume 5757 of *LNCS*, pages 566–577. Springer, 2009.

16 M. Vatshelle. *New Width Parameters of Graphs*. PhD thesis, University of Bergen, Norway, 2012.

Linear Kernels for Outbranching Problems in Sparse Digraphs*

Marthe Bonamy[1], Łukasz Kowalik[2], Michał Pilipczuk[2], and Arkadiusz Socała[2]

1 **LIRMM, France**
marthe.bonamy@lirmm.fr
2 **University of Warsaw, Poland**
kowalik@mimuw.edu.pl, michal.pilipczuk@mimuw.edu.pl, arkadiusz.socala@students.mimuw.edu.pl

Abstract

In the k-Leaf Out-Branching and k-Internal Out-Branching problems we are given a directed graph D with a designated root r and a nonnegative integer k. The question is to determine the existence of an outbranching rooted at r that has at least k leaves, or at least k internal vertices, respectively. Both these problems were intensively studied from the points of view of parameterized complexity and kernelization, and in particular for both of them kernels with $O(k^2)$ vertices are known on general graphs. In this work we show that k-Leaf Out-Branching admits a kernel with $O(k)$ vertices on H-minor-free graphs, for any fixed H, whereas k-Internal Out-Branching admits a kernel with $O(k)$ vertices on any graph class of bounded expansion.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases FPT algorithm, kernelization, outbranching, sparse graphs

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.199

1 Introduction

In this work we are interested in kernelization algorithms for two problems investigated by Dorn et al. [4], namely k-Leaf Out-Branching (LOB) and k-Internal Out-Branching (IOB). In both problems, we are given a directed graph D with a specified root r and a nonnegative integer k. By an *outbranching rooted at* r we mean a spanning tree of D with all the edges oriented away from r. A vertex of D is a *leaf* in an outbranching T if it has outdegree 0 in T, and is *internal* otherwise. In LOB the question is to verify the existence of an outbranching rooted at r that has at least k leaves, whereas in IOB we instead ask for an outbranching rooted at r with at least k internal vertices. Both problems enjoy the existence of kernels with $O(k^2)$ vertices on general graphs [2, 9], however up to this work no better kernels were known even in the case of planar graphs. Although many problems for planar graphs admit kernels of linear size by a general framework of bidimensionality (see [7]), the directed nature of both problems studied here prevents them from satisfying even the most basic properties needed for the bidimensionality tools to be applicable.

* Work partially supported by the ANR Grant EGOS (2012-2015) 12 JS02 002 01 (MB), by the National Science Centre of Poland, grants number 2013/09/B/ST6/03136 (ŁK, AS). This work was done while Michał Pilipczuk has been holding a post-doc position at Warsaw Centre of Mathematics and Computer Science, and has been supported by the Foundation for Polish Science via the START stipend programme.

Dorn et al. [4] designed subexponential parameterized algorithms with running time $2^{\tilde{O}(\sqrt{k})} \cdot n^{O(1)}$ for both problems on H-minor-free graphs[1]. They did it, however, by circumventing in both cases the need of obtaining a linear kernel. In the case of LOB they show how to apply preprocessing rules to obtain an instance that can be still large in terms of k, but has treewidth $O(\sqrt{k})$ so that the dynamic programming on a tree decomposition can be applied. In the case of IOB they apply a variant of Baker's layering technique.

Our results and techniques. In this work we fill the gap left by Dorn et al. [4] and prove that both LOB and IOB admit linear kernels on H-minor-free graphs. In fact, for IOB our approach works even in the more general setting of graph classes of bounded expansion (see Section 2 for a definition). By slightly abusing notation, in what follows we say that a directed graph D belongs to some class of undirected graphs (e.g. is H-minor free) if the underlying undirected graph of D has this property.

▶ **Theorem 1.** *Let H be a fixed graph. There is an algorithm that, given an instance (D, k) of* LOB *where D is H-minor-free, in polynomial time either resolves the instance (D, k), or outputs an equivalent instance (D', k') of* LOB *where $|V(D')| = O(k)$, $k' \leq k$, and D' is H-minor free. The algorithm does not need to know H.*

Note that Theorem 1 implies also a kernel of linear size for any minor-closed family of graphs $\mathcal{G}$. Indeed, by the Roberson and Seymour's graph minor theorem there exists a fixed finite family $\mathcal{H}$ such that $\mathcal{G}$ contains exactly graphs that are H-minor free for every $H \in \mathcal{H}$. By Theorem 1, for any input graph $D \in \mathcal{G}$, the output graph D' is H-minor free for every $H \in \mathcal{H}$. Hence, D' is in $\mathcal{G}$. In particular, it follows that Theorem 1 implies linear kernels for planar graphs and other graphs embeddable on a surface of bounded genus.

▶ **Theorem 2.** *Let $\mathcal{G}$ be a hereditary graph class of bounded expansion. There is an algorithm that, given an instance (D, k) of* IOB *where $D \in \mathcal{G}$, in polynomial time either resolves the instance (D, k), or outputs an equivalent instance (D', k) of* IOB *where $|V(D')| = O(k)$ and D' is an induced subgraph of D.*

By applying these kernelization algorithms and then running dynamic programming on a tree decomposition of the obtained graph, we easily obtain the following corollary.

▶ **Theorem 3.** *Let H be a fixed graph. Then both* LOB *and* IOB *can be solved in time $2^{O(\sqrt{k})} + n^{O(1)}$ when the input is an n-vertex H-minor-free graph.*

Algorithms with a similar running time – but with additional $\log k$ factor in the exponent – were obtained by Dorn et al. [4]. If one follows their approach, then for LOB it is possible to shave off this factor in the exponent just by replacing the dynamic programming on a tree decomposition with a more modern one. However, for IOB the logarithmic factor is caused also by an application of the layering technique, and hence such a replacement and manipulation of parameters in layering would only improve $\log k$ to $\sqrt{\log k}$. By constructing a truly linear kernel we are able to shave this factor completely off. We remark that the running time given by Theorem 3 is optimal under the Exponential Time Hypothesis even on planar graphs; see appendix for further details.

[1] We remark that Dorn et al. state the result for IOB only for apex-minor-free graphs, but a combination of their approach with the contraction decomposition technique of Demaine et al. [3] immediately generalizes the result to H-minor-free graphs.

(a) A weak bipath

(b) A fat bipath (the black vertices may have out-neighbors other than those depicted and some of the vertical edges may be contracted)

Figure 1 Different types of bipaths.

To prove Theorems 1 and 2, we revisit the quadratic kernels on general graphs given by Daligault and Thomassé [2] (for LOB) and by Gutin et al. [9] (for IOB). For LOB we need to modify the approach substantially, as the core reduction rule used by Daligault and Thomassé is the following: whenever there is a *cutvertex* in the graph – a vertex whose removal makes some other vertex not reachable from r – then it is safe to *shortcut* it: remove it and add an arc from every its inneighbor to every its outneighbor. Observe that an application of this rule does not preserve H-minor-freeness, so the kernel of Daligault and Thomassé [2] may start with an H-minor free graph and go outside of this class.

To circumvent this problem, we exploit the structural approach proposed by Dorn et al. [4]. While not achieving a linear kernel in the precise sense, Dorn et al. are able to simplify the structure of the instance so that it fits their purposes. The main idea is to contract cutedges instead of shortcutting cutvertices, which is a weaker operation that, however, preserves H-minor-freeness. Dorn et al. are able to expose a set of so-called *special* vertices S of size linear in k such that $G \setminus S$ has constant pathwidth; this is already enough to employ the bidimensionality technique. To obtain a linear kernel, we need to perform a much more refined analysis of the instance. More precisely, we construct a set S with $|S| = O(k)$ such that $G \setminus S$ is consists of *fat bipaths*: chains as depicted in Figure 1, possibly with some vertical (cut)edges contracted, and with outgoing edges with heads in S. After contracting the vertical edges, such a fat bipath becomes a weak bipath: a bidirectional path possibly with outgoing edges with heads in S. Weak bipaths are crucial in the structural approach of Daligault and Thomassé [2], and our fat bipaths can be thought of as more fuzzy variants of weak bipaths that cannot be reduced due to the inability to shortcut cutvertices.

To obtain a linear kernel, we need to reduce the total length of the fat bipaths. For this, we use concepts borrowed from the analysis of graph classes of bounded expansion, of which H-minor-free classes are special cases. Very recently Drange et al. [5] announced a linear kernel for Dominating Set on graph classes of bounded expansion, and the main tool used there is the analysis of the number of different neighborhoods that can arise in a graph G from a bounded expansion graph class $\mathcal{G}$. Essentially, there is a constant c such that for every $X \subseteq V(G)$ there are only $O(|X|)$ vertices in $V(G) \setminus X$ that neighbor more than c vertices in X, while the vertices of $V(G) \setminus X$ that neighbor at most c vertices in X can be grouped into $O(|X|)$ classes with exactly the same neighborhoods. We apply this idea to the instance at hand with the interior of every fat bipath contracted to one vertex. Thus, we infer that there are only $O(k)$ fat bipaths that neighbor more than c special vertices, and their total length can be bounded by $O(k)$ using reduction rules. On the other hand, fat bipaths with neighborhoods of size at most c are reduced within their neighborhood classes, whose number is also $O(k)$.

The same neighborhood diversity argument plays the key role also in our kernel for IOB (Theorem 2). The idea of Gutin et al. [9] is that if a solution to the instance cannot be found immediately by a simple local search, then one can expose a vertex cover U of size at most $2k$ in the graph. The vertices of $V(D) \setminus U$ are reduced using an argument involving

crown decompositions in an auxiliary graph where vertices of $V(D) \setminus U$ are matched to pairs of adjacent vertices of U; this gives a quadratic dependence on k of the size of the kernel. We observe that in case D belongs to a class of bounded expansion, then there is only $O(|U|) = O(k)$ vertices of $V(D) \setminus U$ that have super-constant neighborhood size in U, while the others are grouped into $O(|U|) = O(k)$ neighborhood classes, each of which can be reduced to constant size using the same approach via crown decompositions.

For IOB we did not need any edge contractions in the reduction rules, so the kernelization procedure works on any graph class of bounded expansion. However, for LOB it seems necessary to apply contractions of subgraphs of unbounded diameter, e.g. to reduce long paths that contribute with at most one leaf to the solution. While the last phase relies mostly on the bounded expansion properties of the graph class, we need to allow contractions in the reduction rules and hence we do not achieve the same level of generality as for IOB.

We see the additional advantage of our approach in its simplicity. Instead of relying on complicated decomposition theorems for H-minor free graphs, which is a standard technique in such a setting, we use the methodology proposed by Drange et al. [5]: To exploit purely combinatorial, abstract notions of sparsity, like the concept of bounded expansion, and in this manner obtain a much cleaner treatment of the considered graph classes. Of particular interest is the usefulness of the approach of grouping vertices according to their neighborhoods in some fixed modulator X, which is the key idea in [5].

Organization of the paper. In Section 2 we give preliminaries on tools borrowed from the analysis of graph classes of bounded expansion. Sections 3 and 4 are devoted to the proofs of Theorems 1 and 2, respectively. Due to space limitations, proofs of claims marked by ★ are skipped in this extended abstract; the reader can find their proofs in our technical report at arxiv [1]. By the same reason, the proof of Theorem 3 and a discussion on the optimality of the obtained algorithms is also skipped.

Notation. In this paper we deal with digraphs. Let $D = (V, E)$ be a digraph. Consider an edge $(u, v) \in E$. We say that v is an *out-neighbor* of u and u is an *in-neighbor* of v. We also say that v is a *head* and u is a *tail* of (u, v). Also, v and u are *neighbors* of each other. For any vertex v we denote the sets of all its neighbors, out-neighbors and in-neighbors by $N_D(v)$, $N_D^+(v)$ and $N_D^-(v)$, respectively. Moreover, the *degree*, *out-degree*, and *in-degree* of v are defined as $\deg_D(v) = |N(v)|$, $\deg_D^+(v) = |N^+(v)|$, and $\deg_D^-(v) = |N^-(v)|$. We omit the subscripts and write simply $N(v)$ or $\deg(v)$ whenever it does not lead to ambiguity. For any set $S \subseteq V$ we denote $N_D^-(S) = \bigcup_{v \in S} N_D^-(v) \setminus S$ and $N_D^+(S) = \bigcup_{v \in S} N_D^+(v) \setminus S$.

2 Preliminaries on Sparse Graphs

In this section we recall some definitions and basic properties of sparse graphs, in particular d-degenerate graphs, bounded expansion graphs and H-minor-free graphs. Although in this section we refer to undirected graphs, all the notions and claims apply also to digraphs, by looking at the underlying undirected graph.

We say that graph G is k-degenerate when every subgraph of G has a vertex of degree at most k. This implies (and in fact is equivalent to) that we can remove all the edges of G by repeatedly removing vertices of degree at most k. It follows that G has at most $k|V(G)|$ edges. The *degeneracy* of a graph is the smallest value of k for which it is k-degenerate. Degeneracy is closely linked to *arboricity*, i.e., minimum number $\mathrm{arb}(G)$ of forests that cover the edges of G: it is well known that degeneracy is between $\mathrm{arb}(G)$ and $2\,\mathrm{arb}(G)$.

Recall that a graph H is a *minor* of graph G if there exists a *minor model* $(I_u)_{u\in V(H)}$ of H in G that satisfies the following properties:

- sets I_u for $u \in V(H)$ are pairwise disjoint subsets of $V(G)$ that moreover induce connected subgraphs;
- for each $uv \in E(H)$, there exist $x_u \in I_u$ and $x_v \in I_v$ such that $x_u x_v \in E(G)$.

For any fixed graph H, the class of *H-minor-free graphs* comprises all the graphs G that do not have H as a minor. Note that H-minor free graphs are closed under minor operations: vertex and edge deletions, and edge contractions. For example, graphs embeddable into a constant genus surface are H-minor-free for some fixed H; in particular, by Kuratowski's theorem, planar graphs are K_5-minor free and $K_{3,3}$-minor free. The following lemma provides a connection between H-minor-free graphs and degeneracy.

▶ **Lemma 4** (see Lemma 4.1 in [10]). *Any H-minor free graph is d_H-degenerate for $d_H = O(|H|\sqrt{\log |H|})$.*

Let r be a nonnegative integer. If a minor model $(I_u)_{u\in V(H)}$ satisfies in addition that $G[I_u]$ has radius at most r for each $u \in V(H)$, then $(I_u)_{u\in V(H)}$ is an *r-shallow minor model of H*, and we say that H is an *r-shallow minor of G*. If $\mathcal{G}$ is a class of graphs, then by $\mathcal{G} \triangledown r$ we denote the class of all r-shallow minors of graphs from $\mathcal{G}$; note that $\mathcal{G} \triangledown 0$ are all subgraphs of graphs of $\mathcal{G}$. We now define the *greatest reduced average degree* (*grad*) of a class $\mathcal{G}$ at depth r as

$$\nabla_r(\mathcal{G}) = \sup_{H\in\mathcal{G}\triangledown r} \frac{|E(H)|}{|V(H)|}.$$

That is, we take the greatest edge density among the r-shallow minors of $\mathcal{G}$. Class $\mathcal{G}$ is said to be of *bounded expansion* if $\nabla_r(\mathcal{G})$ is a finite constant for every r. Observe that then the graphs from $\mathcal{G}$ are in particular d-degenerate for $d = \lfloor 2\nabla_0(\mathcal{G})\rfloor$. For a single graph G, we denote $\nabla_r(G) = \nabla_r(\{G\})$.

Consider the class $\mathcal{G}_H$ of H-minor-free graphs. By Lemma 4, every graph $G \in \mathcal{G}_H$ has at most $d_H \cdot |V(G)|$ edges. Since $\mathcal{G}_H$ is closed under taking minors, it follows that $\mathcal{G}_H \triangledown r = \mathcal{G}_H$ for every nonnegative r, so also $\nabla_r(\mathcal{G}_H) \leq d_H$. Thus, H-minor-free graphs form a class of bounded expansion with all the grads bounded independently of r.

In this paper we do not use the original definition of bounded expansion graphs, but we rather rely on the point of view of diversity of neighborhoods, which was found to be very useful in [5]. More precisely, we now use the following result from [8, Lemma 6.6]; the statement with adjusted notation is taken verbatim from [5].

▶ **Proposition 5** (Proposition 2.5 of [5]). *Let G be a graph, $X \subseteq V(G)$ be a vertex subset, and $R = V(G) \setminus X$. Then for every integer $p \geq \nabla_1(G)$ it holds that*

1. $|\{v \in R\colon |N(v) \cap X| \geq 2p\}| \leq 2p \cdot |X|$, *and*
2. $|\{A \subseteq X\colon |A| < 2p$ *and* $\exists_{v\in R}\ A = N(v) \cap X\}| \leq (4^p + 2p)|X|$.

Consequently, the following bound holds:

$$|\{A \subseteq X\colon \exists_{v\in R}\ A = N(v) \cap X\}| \leq \left(4^{\nabla_1(G)} + 4\nabla_1(G)\right) \cdot |X|.$$

We need a strengthening of the first claim of Proposition 5.

▶ **Lemma 6.** (★) *Let $G = (X, Y, E)$ be a bipartite graph of degeneracy at most d. Then,*

$$\sum_{\substack{y\in Y\\ \deg_G(y)>2d}} \deg_G(y) \leq 2d|X|.$$

Note that Proposition 5 has the following corollary when applied to H-minor-free graphs.

▶ **Corollary 7.** *Let H be a graph. There exists $c_H = 2^{O(|H|\sqrt{\log |H|})}$ such that in any H-minor-free bipartite graph $G = (X, Y, E)$, there are at most $c_H \cdot |X|$ vertices in Y with pairwise distinct neighborhoods in X.*

3 k-Leaf Out-Branching in H-minor-free graphs

In this section we deal with rooted digraphs, i.e., digraphs with a vertex r, called *root*, of in-degree 0. In such digraphs we redefine some standard connectivity notions as follows. Let (D, r) be a rooted digraph. We say that D is *connected* when every vertex of D is reachable from r. A *cut-vertex* is any vertex $v \in V(D) \setminus \{r\}$ such that $D - r$ is not connected. The set of all cut-vertices of D is denoted by $\mathrm{cv}(D)$. We say that D is *2-connected* if D has no cut-vertex (equivalently, for every vertex $v \in V(D) \setminus \{r\}$ there are at least two paths from r to v that do not share internal vertices). Similarly, a *cut-edge* is any edge $(u, v) \in E(D)$ such that $D - (u, v)$ is not connected. We say that D is *2-edge-connected* if D has no cut-edge (equivalently, for every vertex $v \in V(D) \setminus \{r\}$ there are at least two edge-disjoint paths from r to v). Note that if (u, v) is a cut-edge then u is a cut-vertex or $u = r$.

By a *contraction* of edge (a, b) in D we mean the following operation: identify a and b into a newly introduced vertex $v_{(a,b)}$, replace a and b with $v_{(a,b)}$ in every edge of D, and remove all the loops and parallel edges created in this manner. Note that if D is H-minor-free, then it remains H-minor-free after contractions as well.

Following [2], we say that a vertex v of D is *special* if v is of in-degree at least 3 or there is an incoming simple edge, i.e., an edge (u, v) such that $(v, u) \notin E(D)$. The set of all special vertices of D is denoted by $\mathrm{sp}(D)$.

A *weak bipath* P is a sequence of vertices $u_1, \ldots, u_p$ for some $p \geq 3$, such that for each $i = 2, \ldots, p-1$, we have $N^-(u_i) = \{u_{i-1}, u_{i+1}\} \subseteq N^+(u_i)$. The *length* of P is $p - 1$. If additionally $N^+(u_i) = N^-(u_i) = \{u_{i-1}, u_{i+1}\}$ for every $i = 2, \ldots, p-1$, we say that P is *proper bipath* (or shortly a *bipath*). u_1 and u_p are called the *extremities* of P.

We say that a cut-edge (u, v) is *lonely* when there is no other cut-edge with the tail in u. We call a cut-edge *branching* is there is another cut-edge with the same tail. The graph obtained from D by contracting all lonely cut-edges is denoted by D_c and called the *contracted graph*. Consider a vertex v of D_c. Then either v was created by contracting some set of cut-edges Z in D or $v \in D$. In the prior case we define the *bag* B of v as the set of vertices incident to edges in Z. Also, for any edge $(x, y) \in Z$ the vertex x is called a *tail* of B and y is a *head* of B. In the latter case, i.e., when $v \in D$, we define the bag as $B = \{v\}$ and v is both head and tail of B. When B is a bag of v we denote $v_B = v$ and $B_v = B$. If there is exactly one head and exactly one tail of B, then they are denoted by h_B and t_B, respectively. We say that bags A and B are *linked* if there are edges both from A to B and from B to A.

Our kernelization algorithm. Let us describe our algorithm which outputs a kernel for k-Leaf Out-Branching. The algorithm exhaustively applies reduction rules. Each reduction rule is a subroutine which finds in polynomial time a certain structure in the graph and replaces it by another structure, so that the resulting instance is equivalent to the original one. More precisely, we say that a reduction rule for parameterized graph problem P is *correct* when for every instance (D, k) of P it returns an instance (D', k') such that a) (D', k') is an instance of P, b) (D, k) is a yes-instance of P iff (D', k') is a yes-instance of P, and c)

$k' \leq k$. Below we state the rules we use. The rules are applied in the given order, i.e., in each rule we assume that the earlier rules do not apply.

Rule 1. If there exists a vertex not reachable from r in D, then reduce to a trivial no-instance.

Rule 2. If there exists a cut-vertex v with exactly one incoming edge e then contract e. Similarly, if there exists a cut-vertex v with exactly one outgoing edge e then contract e.

Rule 3. Let P be a proper bipath of length 4 in D. Contract any edge of P.

Rule 4. Let x be a vertex of D. If there exists $y \in N^-(x)$ such that the removal of $N^-(x) \setminus \{y\}$ disconnects y from r, then delete the edge (y, x).

The correctness of the above reduction rules was proven in [2]. (In [2], Rule 2 is formulated in a more general way, but we restrict it so that if the input digraph was H-minor-free, then so is the resulting reduced graph.) Let us remark that Rule 4 remains true if $r \in N^-(x) \setminus \{y\}$, and in this case it triggers removal of all the incoming edges apart from the one coming from the root. Below we introduce two simple rules which will make our argument a bit easier.

Rule 5. If there are two cut-edges (x_1, y_1) and (x_2, y_2) such that $(x_1, x_2), (x_2, x_1) \in E(D)$, then contract (x_1, x_2).

Rule 6. If there is a cut-edge (u, v) such that $(v, u) \in E(D)$, then remove (v, u).

▶ **Lemma 8.** (★) *Rules 5 and 6 are correct.*

To complete the algorithm we need a final accepting rule which is applied when the resulting graph is too big. In the remainder of this section we sketch the proof that Rule 7 is correct for H-minor-free graphs for some constant $c = 2^{O(|H|\sqrt{\log |H|})}$.

Rule 7. If the graph has more than $c \cdot k$ vertices, return a trivial yes-instance (conclude that there is a rooted outbranching with at least k leaves in D).

We conclude with the following lemma.

▶ **Lemma 9.** *Let H be a graph. If the input is an H-minor-free graph, then the output of each of the rules 1– 7 is a minor of D, and hence an H-minor-free graph. Moreover, each rule can be recognized and applied in polynomial time, and the degree of the polynomial does not depend on H.*

Proof. The first claim follows from the fact that the rules modify the graph by means of deletions and contractions only. The second claim is straightforward to check. ◀

A few simple properties of the reduced graph. Let us we state simple auxiliary lemmas, which will be used in the remainder of the paper.

▶ **Lemma 10.** (★) *If reduction rules do not apply to D then every bag is of size at most two and contains at most one edge.*

▶ **Lemma 11.** (★) *If reduction rules 1-4 do not apply to D, then for arbitrary pair of bags A and B every edge from A to B has head in t_B.*

▶ **Lemma 12.** (★) *Assume no reduction rule applies to D. Let $S \subseteq V(D_c)$ be any set of vertices that contains the root r and every special vertex of D_c. Then one can find weak bipaths $P_1, P_2, \ldots, P_q$, such that:*

(i) *The sets of internal vertices of $P_1, P_2, \ldots, P_q$ form a partition of $V(D_c) \setminus S$.*

(ii) *The extremities of each P_i belong to S and are distinct.*

(iii) *The out-neighbors of the internal vertices of each P_i belong to S.*

Weak bipaths $P_1, \ldots, P_q$ given by Lemma 12 are called *maximal bipaths*. Note that for every such maximal bipath $P = v_1, v_2, \ldots, v_p$ and every $j = 2, \ldots, p-1$, bag B_{v_j} is linked to $B_{v_{j-1}}$ and $B_{v_{j+1}}$, and to no other bag.

New lower bounds on the number of leaves. Now our goal is to establish a number of lower bounds on the number of leaves. Each of the lower bounds is a linear function of a number of some type of vertices or structures in D. These bounds will help us prove that Rule 7 is correct. Indeed, to this end it suffices to focus on a no-instance and prove that it has at most ck vertices. Hence, if we know that $\mathrm{maxleaf}(D)$ is large when there are many vertices of some kind A, then we know that in our no-instance there are few vertices of kind A. In other words vertices of type A are "easy". In the final part of this section we will show that because of sparsity arguments the number of the remaining vertices (not corresponding to an "easy type") is linear in the number of "easy" vertices. In fact, instead of looking for "easy" vertices in D, we focus on D_c. This is justified by the fact that by Lemma 10 we have $|V(D)| \leq 2|V(D_c)|$, so if we prove that $|V(D_c)| = O(k)$ then also $|V(D)| = O(k)$.

Daligault and Thomassé [2] show the following lower bound.

▶ **Theorem 13** ([2]). *Let D be a 2-connected rooted digraph. Then* $\mathrm{maxleaf}(D) \geq \frac{|\mathrm{sp}(D)|}{30}$.

Unfortunately, D_c is not necessarily 2-connected so we cannot use the above bound. However, we can generalize Theorem 13 as follows.

▶ **Theorem 14.** (★) *Let D be a connected rooted digraph such that every cut-edge is branching. Then* $\mathrm{maxleaf}(D) \geq \frac{|\mathrm{sp}(D)|}{30} - \mathrm{cv}(D)$ *and* $\mathrm{maxleaf}(D) \geq \frac{|\mathrm{sp}(D)|}{60}$.

We are able to show that in D_c every cut-edge is branching and $\mathrm{maxleaf}(D) \geq \mathrm{maxleaf}(D_c)$ (proofs skipped in this extended abstract). This implies the following.

▶ **Lemma 15.** (★) *Assume that rules 1-6 do not apply to D. Then,* $\mathrm{maxleaf}(D) \geq \frac{|\mathrm{sp}(D_c)|}{60}$.

We say that a bag B is *special* when v_B is special in D_c. We say that a bag B is *isolated* when B is a non-special bag of size 2 and there is no edge from t_B to a special bag. Vertex $v \in V(D_c)$ is *isolated* if $v = v_B$ for some isolated bag B. The set of all isolated vertices in D_c is denoted by $\mathrm{iso}(D_c)$.

▶ **Lemma 16.** (★) *If reduction rules do not apply to D then* $\mathrm{maxleaf}(D) \geq \frac{|\mathrm{iso}(D_c)|}{180}$.

We will say that a vertex v of D_c is *easy* when $v = r$, or v is special, or v is isolated in D_c. A vertex that is not easy is called *hard*. We now invoke Lemma 12 for S being the set of all the easy vertices. Every maximal bipath obtained in this decomposition will be called a *maximal hard bipath*. In other words, a weak bipath in D_c is *hard* if all its internal vertices are hard. The sets of all easy and hard vertices in D_c are denoted by $\mathrm{ea}(D_c)$ and $\mathrm{hd}(D_c)$, respectively. For any maximal hard weak bipath P' in D_c define $O(P') = N^+_{D_c}(V(P') \setminus \{u, v\})$, where u and v are the extremities of P'.

For every pair of easy vertices $u, v \in \mathrm{ea}(D_c)$ and a subset $S \subseteq V(D_c)$ with $\{u, v\} \subseteq S$, if there is a hard bipath P' between u and v such that $O(P') = S$, we choose arbitrarily two such paths (or one, if only one exists) and we call them *masters*, while all the remaining hard bipaths P'' between u and v with $O(P'') = S$ are called *slaves* of respective masters, or just *slaves*. The number of all slaves in D_c is denoted by $\mathrm{sl}(D_c)$.

▶ **Lemma 17.** (★) $\mathrm{maxleaf}(D) \geq \mathrm{sl}(D_c)$.

The size bound. The following theorem implies the correctness of Rule 7.

▶ **Theorem 18.** *Let H be a graph. Let D be an H-minor-free digraph such that rules 1–6 do not apply. If* $\mathrm{maxleaf}(D) < k$, *then* $|V(D)| = 2^{O(|H|\sqrt{\log |H|})}k$.

In what follows we prove Theorem 18. We assume that rules 1–6 do not apply to D. Since $\mathrm{maxleaf}(D) < k$, our lower bounds on $\mathrm{maxleaf}(D)$ imply an upper bound of $O(k)$ on the number of easy vertices. Our plan now is to show a linear bound on the number of hard vertices in terms of $|\mathrm{ea}(D_c)| + \mathrm{sl}(D_c)$ and next get a bound on $|V(D)|$ as a corollary. To this end, we state a few useful properties of hard weak bipaths in D_c.

▶ **Lemma 19.** *Let $\ell \geq 9$ and let $P' = v_1, \ldots, v_\ell$ be a hard bipath in D_c such that v_1 and v_ℓ are easy. For every $i = 3, \ldots, \ell - 6$ there is at least one edge in D from $t_{B_{v_j}}$, for some $j = i, \ldots, i + 4$, to a vertex outside $\cup_{j'=2}^{\ell-1} B_{v_{j'}}$.*

Proof. Fix $i \in \{3, \ldots, \ell - 6\}$ and consider the length 4 bipath $v_i, \ldots, v_{i+4}$. Denote $B_j = B_{v_j}$. If for some $j = i + 1, i + 2, i + 3$ there is an edge from B_j with head $h \notin B_{j-1} \cup B_{j+1}$, then by Lemma 12(*iii*), $h \notin \cup_{j'=2}^{\ell-1} B_{v_{j'}}$ and we are done. Hence the edges leaving B_{i+1}, B_{i+2}, and B_{i+3} go only to the neighboring bags. Since Rule 3 does not apply, for some $j = i, \ldots, i + 4$ the bag B_j is of size 2. Since v_j is hard, B_j is not isolated. Hence, there is an edge e in D from t_{B_j} to a special bag B. Since $v_2, \ldots, v_{\ell-1}$ are hard, B is none of $B_2, \ldots, B_{\ell-1}$. ◀

▶ **Lemma 20.** *For any maximal hard weak bipath P' in D_c,* $|\mathrm{hd}(D_c) \cap V(P')| \leq 10|O(P')| + 6$.

Proof. Let $P' = v_1, \ldots, v_\ell$. We can assume that $\ell \geq 9$, for otherwise $|\mathrm{hd}(D_c) \cap V(P')| \leq 6$ and the claim holds trivially. For convenience denote $B_i = B_{v_i}$. By Lemma 19 there are at least $\lfloor \frac{\ell-4}{5} \rfloor$ edges from tails of bags $B_3, \ldots, B_{\ell-2}$ to vertices outside $\cup_{i=2}^{\ell-1} B_{v_j}$. Let Z denote the set of these edges. We claim that for every vertex $u \in V(D)$ there are at most two edges from Z with heads in u. Indeed, assume that u has got three in-neighbors $t_{B_a}, t_{B_b}, t_{B_c}$ in D, with $a < b < c$. Then $N^-(u) \setminus \{t_{B_b}\}$ cuts t_{B_b} (and all vertices of $B_{a+1}, \ldots, B_{c-1}$) from r, a contradiction to the fact that D is reduced with respect to Rule 4. Hence the edges in Z have at least $\lfloor \frac{\ell-4}{5} \rfloor \cdot \frac{1}{2} \geq \frac{\ell-8}{5} \cdot \frac{1}{2}$ different heads. By Lemma 11 these heads are tails of bags, and by Lemma 10 each of them corresponds to a different vertex in D_c. It follows that the vertices $v_3, \ldots, v_{\ell-2}$ have in D_c at least $\frac{\ell-8}{10}$ neighbors in $O(P')$, so $|O(P')| \geq \frac{\ell-8}{10}$. Since $|\mathrm{hd}(D_c) \cap V(P)| = \ell - 2$ it follows that $|\mathrm{hd}(D_c) \cap V(P)| \leq 10|O(P')| + 6$. ◀

In what follows we are going to bound the size of D_c using its sparsity properties. To this end we use an auxiliary bipartite graph G, called the *bipath minor* of D_c, constructed as follows. We put $V(G) = A \cup B$, where $A = \mathrm{ea}(D_c)$, and B is the set of all maximal hard bipaths in D_c. For every maximal hard bipath P' in D_c with extremities $u, v \in \mathrm{ea}(D_c)$, the neighborhood of the corresponding vertex in B is exactly $O(P)$.

▶ **Lemma 21.** *If D is H-minor-free, then* $|\mathrm{hd}(D_c)| = 2^{O(|H|\sqrt{\log |H|})}(|\mathrm{ea}(D_c)| + \mathrm{sl}(D_c))$.

Proof. Consider an arbitrary hard vertex v of D_c. Consider the maximal hard weak bipath P' in D_c that contains v. Then P' corresponds to a vertex in B and by Lemma 20, it has at most $10|O(P')| + 6$ internal vertices. It follows that

$$|\mathrm{hd}(D_c)| \leq \sum_{v \in B} (10 \deg_G(v) + 6) \leq \sum_{v \in B} 16 \deg_G(v). \tag{1}$$

Note that G is a minor of (the undirected version of) D since it can be obtained from D_c by edge contractions and deletions, and D_c in turn is obtained from D by contractions.

Hence, G is H-minor-free. Moreover, G is simple. By Lemma 4, we know that G is d_H-degenerate, for $d_H = O(|H|\sqrt{\log|H|})$. Let B_m and B_s denote the vertices in B for which the corresponding maximal hard bipath is master and slave, respectively. By (1) we get

$$|\mathrm{hd}(D_c)| \leq 16 \sum_{v\in B} \deg_G(v) \leq 16 \sum_{\substack{v\in B\\ \deg_G(v)>2d_H}} \deg_G(v) + 16 \sum_{\substack{v\in B_s\\ \deg_G(v)\leq 2d_H}} \deg_G(v) + 16 \sum_{\substack{v\in B_m\\ \deg_G(v)\leq 2d_H}} \deg_G(v)$$

Let us bound each of the terms separately. By Lemma 6, we have

$$\sum_{\substack{v\in B_s\\ \deg_G(v)>2d_H}} d(v) \leq 2d_H|A| = O(|H|\sqrt{\log|H|}\cdot|\mathrm{ea}(D_c)|).$$

Obviously,

$$\sum_{\substack{v\in B_s\\ \deg_G(v)\leq 2d_H}} \deg_G(v) \leq 2d_H\mathrm{sl}(D_c) = O(|H|\sqrt{\log|H|}\mathrm{sl}(D_c)).$$

Finally,

$$\sum_{\substack{v\in B_m\\ \deg_G(v)\leq 2d_H}} \deg_G(v) = \sum_{\substack{S\subseteq A\\ |S|\leq 2d_H}} |S|\cdot|\{v\in B_m\colon N_G(v)=S\}| \leq 2d_H \sum_{\substack{S\subseteq A\\ |S|\leq 2d_H}} |\{v\in B_m\colon N_G(v)=S\}|.$$

By Corollary 7, there is a constant $c_H = 2^{O(|H|\sqrt{\log|H|})}$ such that there are at most $c_H|A|$ distinct neighborhoods of vertices in B. For each such neighborhood $S\subseteq A$ and for every pair of vertices $u,v\in S$ there are at most two master bipaths P' with endpoints u and v and such that $O(P')=S$. Therefore for a fixed neighborhood S of size at most $2d_H$ we have $|\{v\in B_m|N_G(v)=S\}| \leq 2\binom{|S|}{2} \leq 2\binom{2d_H}{2} = O(d_H^2)$. Hence

$$\sum_{S\subseteq A,|S|\leq 2d_H} |\{v\in B_m|N_G(v)=S\}| = O(c_H\cdot d_H^2\cdot|\mathrm{ea}(D_c)|) = 2^{O(|H|\sqrt{\log|H|})}|\mathrm{ea}(D_c)|.$$

The claim follows. ◀

Now we can finish the proof of Theorem 18. Assume $\mathrm{maxleaf}(D) < k$. By Lemmas 15 and 16, $\mathrm{ea}(D_c) < 60k + 180k$. Moreover, by Lemma 17, $\mathrm{sl}(D_c) < k$. This, with Lemma 21 gives the claim of Theorem 18.

4 k-internal Out-Branching in graphs of bounded expansion

In this section we give a linear kernel for IOB on any graph class $\mathcal{G}$ of bounded expansion. To this end, we modify the approach of Gutin, Razgon and Kim [9]. Before we proceed to the argumentation, let us remark that Gutin et al. work with a slightly more general problem, where the root of the outbranching is not prescribed; of course, the outbranching is still required to span the whole vertex set. Note that the variant with a prescribed root r can be reduced to this variant simply by removing all in-arcs of r, which forces r to be the root of any outbranching of the given digraph. Since our kernel will be an induced subgraph of D and r will not be removed by any reduction, it will be still true that r is the only candidate for the root of an outbranching. Hence, the resulting instance will be equivalent in both variants. Therefore, from now on we work with variant without prescribed root in order to be able to use the observations of Gutin et al. as black-boxes.

First, Gutin et al. observe that in an instance that cannot be easily resolved, one can find a small vertex cover (of the underlying undirected graph).

▶ **Lemma 22** ([9]). *Given a digraph D, we can either build an out-branching with at least k internal vertices or obtain a vertex cover of size at most $2k-2$ in $O(n^2m)$ time.*

For a given directed graph D and a vertex cover U in D we build an undirected bipartite graph $B_{D,U}$ as follows. Let $W = V(D) \setminus U$. Then,

$$\begin{aligned} V(B) &= U' \cup W, \text{ where } U' = N^-(W) \cup (U \times U); \\ E(B) &= \{\{xy, w\} \ : \ xy \in U \times U,\ w \in W, (x,w) \in E(D),\ (w,y) \in E(D)\} \cup \\ &\quad \{\{x, w\} \ : \ x \in U,\ w \in W,\ (x,w) \in E(D)\}. \end{aligned}$$

A *crown decomposition* of an undirected graph G is a partitioning of $V(G)$ into three parts C, H and R, such that

- C is an independent set.
- There are no edges between vertices of C and R. That is, H separates C and R.
- C can be partitioned into $C_m \cup C_u$ with $|C_m| = |H|$, such that $G[C_m \cup H]$ contains a perfect matching that matches each vertex of C_m with a vertex of H.

Crown decompositions are used in multiple kernelization algorithms. In particular, the following lemma, which Gutin et al. attribute to Fellows et al. [6], shows that in certain situations a crown decomposition can be found efficiently.

▶ **Lemma 23** (see [9]). *Suppose G is an undirected graph on n vertices, and suppose I is an independent set in G such that $|I| \geq \frac{2n}{3}$. Then G admits a crown decomposition $(C = C_u \uplus C_m, H, R)$ with $C \subseteq I$, $H \subseteq V(G) \setminus I$ and $C_u \neq \emptyset$. Moreover, given I, the decomposition $(C = C_u \uplus C_m, H, R)$ can be found in $O(nm)$ time.*

The main idea of Gutin et al. is to search for crowns in $B_{D,U}$ with $C \subseteq W$ and $C_u \neq \emptyset$. Such crowns can be conveniently reduced using the following reduction rule, whose correctness is proved in Lemma 4.4 of [9].

Rule 1. Let U be a vertex cover in D and let $W = V(D) \setminus U$. Assume there is a crown decomposition $(C = C_m \cup C_u, H, R)$ in $B_{D,U}$ with $C \subseteq W$ and $C_u \neq \emptyset$. Then remove C_u from D.

Our idea is to combine Rule 1 with the knowledge that D belongs to a graph class of bounded expansion $\mathcal{G}$, and hence Proposition 5 can be used to reason about the sparseness of the adjacency structure between U and W. Let us introduce some notation. Consider a vertex cover U and an independent set $W = V(D) \setminus U$ in D. Let $W_s = \{w \in W \ : \ \deg_D(w) < 2\nabla_0(\mathcal{G})\}$, and let $W_b = W \setminus W_s$. Moreover, for $N \subseteq U$ with $|N| < 2\nabla_0(\mathcal{G})$, let $W_N = \{w \in W_s \ : \ N(w) = N\}$. Let $\mathcal{N}(U) = \{N \subseteq U \ : \ |N| < 2\nabla_0(\mathcal{G}), W_N \neq \emptyset\}$. Note that $|\mathcal{N}(U)| \leq |W_s|$. Our kernelization algorithm is as follows.

1. If the algorithm from Lemma 22 returns an outbranching, answer YES and terminate; otherwise it returns a vertex cover U of size at most $2k-2$. Let $W = V(D) \setminus U$.
2. Construct the graph $B := B_{D,U}$ and compute W_s, $\mathcal{N}(U)$, and nonempty sets W_N.
3. If there is a set $N \in \mathcal{N}(U)$ such that $|W_N| > 2|N_B(W_N)|$, then apply Lemma 23 to graph $B[N_B[W_N]]$ with $I = W_N$. This gives us a crown decomposition $(C = C_u \uplus C_m, H, R)$ of $B[N_B[W_N]]$ with $C \subseteq W_N$, $H \subseteq N_B(W_N)$, and $C_u \neq \emptyset$. Observe that $(C = C_u \uplus C_m, H, R \cup (V(B) \setminus N_B[W_N]))$ is a crown decomposition of B. Apply Rule 1 to this crown decomposition in order to remove C_u from D, and restart the algorithm in the reduced graph.
4. Otherwise, return D.

In case we have a prescribed root r of the outbranching that we would like to preserve in the kernelization process, we can add it to the constructed vertex cover U, thus increasing its size up to at most $2k-1$. The reduction rules never remove any vertex of U.

Given this algorithm, we can restate and prove our main result for IOB.

▶ **Theorem 2.** *Let $\mathcal{G}$ be a hereditary graph class of bounded expansion. There is an algorithm that, given an instance (D,k) of* IOB *where $D \in \mathcal{G}$, in polynomial time either resolves the instance (D,k), or outputs an equivalent instance (D',k) of* IOB *where $|V(D')| = O(k)$ and D' is an induced subgraph of D.*

Proof. The correctness of our kernelization algorithm and a polynomial bound on its running time follows from Lemmas 22 and 23. Note that the kernelization algorithm never decrements the budget k, so it suffices to show that it outputs an instance (D,k) such that $|V(D)| = O(k)$.

We can assume that the algorithm constructed a vertex cover U of D of size at most $2k-2$ ($2k-1$ if we want to preserve a prescribed root), because otherwise the algorithm would terminate and provide a positive answer. Let $W = V(D) \setminus U$. Then $V(D) = U \cup W_s \cup W_b$. By the first claim of Proposition 5 we get $|W_b| \leq 2\nabla_0(G)|U| \leq 4\nabla_0(\mathcal{G})k$. Hence it suffices to bound the size of W_s. Note that $W_s = \bigcup_{N \in \mathcal{N}(U)} W_N$. By the second claim of Proposition 5 we get $|\mathcal{N}(U)| \leq (4^{\nabla_1(G)} + 2\nabla_1(G))|U| = O(4^{\nabla_1(\mathcal{G})}k)$. However, since Step 2 of the kernelization algorithm cannot be applied, for every $N \in \mathcal{N}(U)$ we have $|W_N| \leq 2|N_{B_{D,U}}(W_N)|$. However, by the construction of $B_{D,U}$ it is clear that $|N_{B_{D,U}}(W_N)| \leq |N|^2 + |N| < 4\nabla_0(\mathcal{G})^2 + 2\nabla_0(\mathcal{G})$, and hence $|W_N| < 8\nabla_0(\mathcal{G})^2 + 4\nabla_0(\mathcal{G})$. It follows that $|W_s| = \sum_{N \in \mathcal{N}(U)} |W_N| = O(4^{\nabla_1(\mathcal{G})}\nabla_0(\mathcal{G})^2 k)$, and hence $|V(D)| = |U| + |W_s| + |W_b| = O(4^{\nabla_1(\mathcal{G})}\nabla_0(\mathcal{G})^2 k)$. This finishes the proof. ◀

Acknowledgments. The authors are very grateful to Marcin Pilipczuk for reading the manuscript carefully and providing useful comments.

References

1 Marthe Bonamy, Łukasz Kowalik, Michał Pilipczuk, and Arkadiusz Socała. Linear kernels for outbranching problems in sparse digraphs. *CoRR*, abs/1509.01675, 2015.

2 Jean Daligault and Stéphan Thomassé. On finding directed trees with many leaves. In *Parameterized and Exact Computation*, pages 86–97. Springer, 2009.

3 Erik D. Demaine, MohammadTaghi Hajiaghayi, and Ken-ichi Kawarabayashi. Contraction decomposition in H-minor-free graphs and algorithmic applications. In *Proc. STOC'11*, pages 441–450. ACM, 2011.

4 Frederic Dorn, Fedor V. Fomin, Daniel Lokshtanov, Venkatesh Raman, and Saket Saurabh. Beyond bidimensionality: Parameterized subexponential algorithms on directed graphs. *Inf. Comput.*, 233:60–70, 2013.

5 Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Stephan Kreutzer, Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, Felix Reidl, Saket Saurabh, Fernando Sánchez Villaamil, and Somnath Sikdar. Kernelization and sparseness: the case of dominating set. *CoRR*, abs/1411.4575, 2014.

6 Mike Fellows, Pinar Heggernes, Frances A. Rosamond, Christian Sloper, and Jan Arne Telle. Finding k disjoint triangles in an arbitrary graph. In *WG'04*, pages 235–244, 2004.

7 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Bidimensionality and kernels. In *Proc. SODA'10*, pages 503–510, 2010.

8 Jakub Gajarský, Petr Hlinený, Jan Obdrzálek, Sebastian Ordyniak, Felix Reidl, Peter Rossmanith, Fernando Sanchez Villaamil, and Somnath Sikdar. Kernelization using structural parameters on sparse graph classes. In *ESA 2013*, pages 529–540. Springer, 2013.

9 Gregory Gutin, Igor Razgon, and Eun Jung Kim. Minimum leaf out-branching and related problems. *Theor. Comput. Sci.*, 410(45):4571–4579, 2009.
10 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity: Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and Combinatorics*. Springer, 2012.

Maximum Matching Width: New Characterizations and a Fast Algorithm for Dominating Set

Jisu Jeong[1], Sigve Hortemo Sæther[2], and Jan Arne Telle[3]

1 **Department of Mathematical Sciences, KAIST, 291 Daehak-ro Yuseong-gu Daejeon, South Korea***
jjisu@kaist.ac.kr

2,3 **Department of Informatics, University of Bergen, Bergen, Norway**
sigve.sether@ii.uib.no, telle@ii.uib.no

Abstract

We give alternative definitions for maximum matching width, e.g. a graph G has $\mathrm{mmw}(G) \leq k$ if and only if it is a subgraph of a chordal graph H and for every maximal clique X of H there exists $A, B, C \subseteq X$ with $A \cup B \cup C = X$ and $|A|, |B|, |C| \leq k$ such that any subset of X that is a minimal separator of H is a subset of either A, B or C. Treewidth and branchwidth have alternative definitions through intersections of subtrees, where treewidth focuses on nodes and branchwidth focuses on edges. We show that mm-width combines both aspects, focusing on nodes and on edges. Based on this we prove that given a graph G and a branch decomposition of mm-width k we can solve Dominating Set in time $O^*(8^k)$, thereby beating $O^*(3^{\mathrm{tw}(G)})$ whenever $\mathrm{tw}(G) > \log_3 8 \times k \approx 1.893k$. Note that $\mathrm{mmw}(G) \leq \mathrm{tw}(G)+1 \leq 3\,\mathrm{mmw}(G)$ and these inequalities are tight. Given only the graph G and using the best known algorithms to find decompositions, maximum matching width will be better for solving Dominating Set whenever $\mathrm{tw}(G) > 1.549 \times \mathrm{mmw}(G)$.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases FPT algorithms, treewidth, dominating set

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.212

1 Introduction

The treewidth $\mathrm{tw}(G)$ and branchwidth $\mathrm{bw}(G)$ of a graph G are connectivity parameters of importance in algorithm design. By dynamic programming along the associated tree decomposition or branch decomposition one can solve many graph optimization problems in time linear in the graph size and exponential in the parameter. For any graph G, its treewidth and branchwidth are related by $\mathrm{bw}(G) \leq \mathrm{tw}(G) + 1 \leq \frac{3}{2}\,\mathrm{bw}(G)$ [15]. The two parameters are thus equivalent with respect to fixed parameter tractability (FPT), with a problem being FPT parameterized by treewidth if and only if it is FPT parameterized by branchwidth. For some of these problems the best known FPT algorithms are optimal, up to some complexity theoretic assumption. For example, Minimum Dominating Set Problem can be solved in time $O^*(3^{\mathrm{tw}(G)})$ when given a decomposition of treewidth $\mathrm{tw}(G)$ [17] but not in time $O^*((3-\varepsilon)^{\mathrm{tw}(G)})$ for any $\varepsilon > 0$ unless the Strong Exponential Time Hypothesis (SETH) fails [12].

* The first author is supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (2011-0011653).

Recently, a graph parameter equivalent to treewidth and branchwidth was introduced, the maximum matching width (or mm-width) $\mathrm{mmw}(G)$, defined by a branch decomposition over the vertex set of the graph, using the symmetric submodular cut function obtained by taking the size of a maximum matching of the bipartite graph crossing the cut (by König's Theorem equivalent to minimum vertex cover) [18]. For any graph G we have $\mathrm{mmw}(G) \leq \mathrm{bw}(G) \leq \mathrm{tw}(G)+1 \leq 3\,\mathrm{mmw}(G)$ and these inequalities are tight, for example any balanced decomposition tree will show that $\mathrm{mmw}(K_n) = \lceil \frac{n}{3} \rceil$.

In this paper we show that given a branch decomposition over the vertex set of mm-width k we can solve Dominating Set in time $O^*(8^k)$. This runtime beats the $O^*(3^{\mathrm{tw}(G)})$ algorithm for treewidth [17] whenever $\mathrm{tw}(G) > \log_3 8 \times k \approx 1.893k$. If we assume only G as input, then since mm-width has a submodular cut function [16] we can approximate mm-width to within a factor $3\,\mathrm{mmw}(G)+1$ in $O^*(2^{3\,\mathrm{mmw}(G)})$ time using the generic algorithm of [13], giving a total runtime for solving dominating set of $O^*(2^{9\,\mathrm{mmw}(G)})$. For treewidth we can in $O^*(2^{3.7\,\mathrm{tw}(G)})$ time [1] get an approximation to within a factor $(3+2/3)\,\mathrm{tw}(G)$ giving a total runtime for solving dominating set of $O^*(3^{3.666\,\mathrm{tw}(G)})$.[1] This implies that on input G, using maximum matching width gives better exponential factors whenever $\mathrm{tw}(G) > 1.549\,\mathrm{mmw}(G)$.

Our results are based on a new characterization of graphs of mm-width at most k, as intersection graphs of subtrees of a tree. It can be formulated as follows, encompassing analogous formulations for all three parameters mm-width (respectively treewidth, respectively branchwidth):

For any $k \geq 2$ a graph G on vertices $v_1, v_2, ..., v_n$ has $\mathrm{mmw}(G) \leq k$ (resp. $\mathrm{tw}(G) \leq k-1$, resp. $\mathrm{bw}(G) \leq k$) if and only if there exist a tree T of max degree at most 3 with nontrivial subtrees $T_1, T_2, ..., T_n$ such that if $v_iv_j \in E(G)$ then subtrees T_i and T_j have at least one node (resp. node, resp. edge) of T in common and for each edge (resp. node, resp. edge) of T there are at most k subtrees using it.

Thus, while treewidth has a focus on nodes and branchwidth a focus on edges, mm-width combines the aspects of both. We also arrive at the following alternative characterization: a graph G has $\mathrm{mmw}(G) \leq k$ if and only if it is a subgraph of a chordal graph H and for every maximal clique X of H there exists $A, B, C \subseteq X$ with $A \cup B \cup C = X$ and $|A|, |B|, |C| \leq k$ such that any subset of X that is a minimal separator of H is a subset of either A, B or C. In fact, using techniques introduced by Bodlaender and Kloks [4] these new characterizations will also allow us to compute a branch decomposition of optimal mm-width in FPT time [9]. In Section 2 we give definitions. In Section 3 we define unique minimum vertex covers for any bipartite graph, show some monotonicity properties of these, and use this properties to give the new characterizations of mm-width. In Section 4 we give the dynamic programming algorithm for dominating set. We end in Section 5 with some discussions.

2 Definitions

For a simple and loopless graph $G = (V, E)$ and its vertex v, let $N(v)$ be the set of all vertices adjacent to v in G, and $N[v] = N(v) \cup \{v\}$. For a subset S of $V(G)$, let $N(S)$ be the set of all vertices that are not in S but are adjacent to some vertex of S in G, and $N[S] = N(S) \cup S$.

A *tree decomposition* of a graph G is a pair $(T, \{X_t\}_{t \in V(T)})$ consisting of a tree T and a family $\{X_t\}_{t \in V(T)}$ of vertex sets $X_t \subseteq V(G)$, called *bags*, satisfying the following three

[1] Note that there is also an $O^*(c^{\mathrm{tw}(G)})$ time 3-approximation of treewidth [3], but the c is so large that the approximation alone has a bigger exponential part than the entire Dominating Set algorithm when using the 3.666-approximation.

conditions:

1. each vertex of G is in at least one bag,
2. for each edge uv of G, there exists a bag that contains both u and v, and
3. for vertices u, v, w of T, if v is on the path from u to w, then $X_u \cap X_w \subseteq X_v$.

The *width* of a tree decomposition $(T, \{X_t\}_{t \in V(T)})$ is $\max_{t \in V(T)} |X_t| - 1$. The *treewidth* of G, denoted by $\mathrm{tw}(G)$, is the minimum width over all possible tree decompositions of G.

A *branch decomposition* over X, for some set of elements X, is a pair (T, δ), where T is a tree over vertices of degree at most 3, and δ is a bijection from the leaves of T to the elements in X. Any edge ab disconnects T into two subtrees T_a and T_b. Likewise, any edge ab partitions the elements of X into two parts A and B, namely the elements mapped by δ from the leaves of T_a, and of T_b, respectively. An edge $ab \in E(T)$ is said to *induce* the partition (A, B).

A *rooted branch decomposition* is a branch decomposition (T, δ) where we subdivide an edge of T and make the new vertex the root r. In a rooted branch decomposition, for an internal vertex $v \in V(T)$, we denote by $\delta(v)$ the union of $\delta(l)$ for all leaves of l having v as its ancestor.

Given a symmetric ($f(A) = f(\overline{A})$) function $f : 2^X \to \mathbb{R}$, using branch decompositions over X, we get a nice way of defining width parameters: For a branch decomposition (T, δ) and edge $e \in T$, we define the *f-value* of the edge e to be the value $f(A) = f(B)$ where A and B are the two parts of the partition induced by e in (T, δ), denoted $f(e)$. We define the *f-width* of branch decomposition (T, δ) to be the maximum f-value over all edges of T, denoted $f(T, \delta)$: $\max_{e \in T}\{f\text{-value of } e\}$. For set X of elements, we define the *f-width* of X to be the minimum f-width over all branch decompositions over X. If $|X| \leq 1$, then X admits no branch decomposition and we define its f-width to be $f(\emptyset)$.

For a graph G and a subset $S \subseteq E(G)$, the *branchwidth* $\mathrm{bw}(G)$ of G is the f-width of $E(G)$ where $f : 2^{E(G)} \to \mathbb{R}$ is a function such that $f(S)$ is the number of vertices that are incident with an edge in S as well as an edge in $E(G) \setminus S$.

The *Maximum Matching-width* of a graph G, mm-width in short, is a width parameter defined through branch decompositions over $V(G)$ and the cardinality of matchings. For a subset $S \subseteq V(G)$, the Maximum Matching-value is defined to be the size of a maximum matching in $G[S, V(G) \setminus S]$, denoted $\mathrm{mm}(S)$. The mm-width of a graph G, denoted $\mathrm{mmw}(G)$, is the f-width of $V(G)$ for $f = \mathrm{mm}$.

3 Subtrees of a tree representation for mm-width

3.1 König covers

In this subsection, we will define canonical minimum vertex covers for any bipartite graph. Our starting point is a well-known result in graph theory.

▶ **Theorem 1** (König's Theorem [10]). *Given a bipartite graph G, for any maximum matching M and minimum vertex cover C of G, the number of edges in M is the same as the number of vertices in C; $|M| = |C|$.*

Let (A, B) be the vertex partition of G. This statement can be proved in multiple ways. The harder direction, that a maximum matching is never smaller than a minimum vertex cover, does not hold for general graphs, and is usually proven by taking a maximum matching M and constructing a vertex cover C having size exactly $|M|$, as follows:

> For each edge $ab \in M$ (where $a \in A$, and $b \in B$), if ab is part of an alternating path starting in an unsaturated vertex of A, then put b into C, otherwise put a into C.

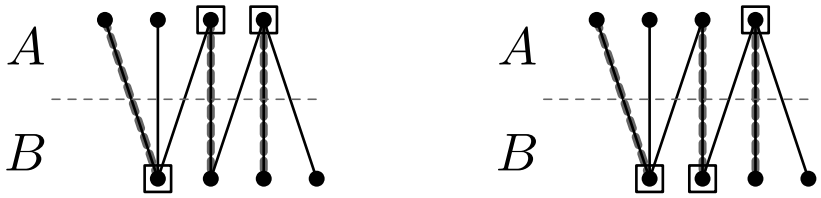


Figure 1 A-König cover and B-König cover.

For a proof that C indeed is a minimum vertex cover of G, see e.g. [7]. We will call the vertex cover C constructed by the above procedure the *A-König cover* of G. A B-König cover of G is constructed similarly by changing the roles of A and B (see Figure 1).

Lemma 2 below shows that the A-König cover will, on the A-side consist of the A-vertices in the union over all minimum vertex covers, and on the B-side will consist of the B-vertices in the intersection over all minimum vertex covers.

▶ **Lemma 2.** *For a bipartite graph $G = (A \cup B, E)$ and minimum vertex cover C of G, the set C is the A-König cover of G if and only if for any minimum vertex cover C' of G we have $A \cap C' \subseteq A \cap C$, and $B \cap C' \supseteq B \cap C$.*

Proof. Let M be a maximum matching of G, and C^* the A-König cover of G constructed from M. Since both C^* and C are minimum vertex covers, by showing that for any minimum vertex cover C' of G we have $A \cap C' \subseteq A \cap C^*$, and $B \cap C' \supseteq B \cap C^*$, as a consequence will also show that $C' = C^*$ if and only if for all minimum vertex covers C' of G we have $A \cap C' \subseteq A \cap C$ and $B \cap C' \supseteq B \cap C$. So this is precisely what we will do.

Let C' be any minimum vertex cover, and b any vertex in $C^* \cap B$. We will show that $b \in C'$, and from that conclude $B \cap C' \supseteq B \cap C^*$. As $b \in C^*$ there must be some alternating path from b to an unsaturated vertex $u \in A$. The vertices b and u are on different sides of the bipartite graph, so the alternating path P between u and b must be of some odd length $2k+1$. From Theorem 1, we deduce that one and only one endpoint of each edge in M must be in C'. As each vertex in $V(P)$ is incident with at most two edges of P, and all edges of P must be covered by C', we need at least $\lceil (2k+1)/2 \rceil = k+1$ of the vertices in $V(P)$ to be in C'. However, the vertices of $V(P) - b$ are incident with only k edges of M. Therefore at most k of the vertices $V(P) - b$ can be in C'. In order to have at least $k+1$ vertices from $V(P)$ in C' we thus must have $b \in C'$.

We now show that $C' \cap A \subseteq C^* \cap A$ by showing that $a \in C^*$ if $a \in A \cap C'$. Let E^* and E' be the edges of G not covered by $C^* \cap B$ and $C' \cap B$, respectively. Since $C^* \cap B \subseteq C' \cap B$, the set E^* must contain all the edges of E'. As C' is a minimum vertex cover, and all edges other than E' are covered by $C' \cap B$, a vertex a of A is in C' only if it covers an edge $e \in E'$. As $E' \subseteq E^*$, we have $e \in E^*$, and hence C^* must also cover e by a vertex in A. As G is bipartite, the only vertex from A that covers e is a, and we can conclude that $a \in C^*$. ◀

The following lemma establishes an important monotonicity property for A-König covers. For a set S of vertices and a vertex v, denote $S + v = S \cup \{v\}$.

▶ **Lemma 3.** *Given a graph G and tripartition (A, B, X) of the vertices $V(G)$, the following two properties holds for the A-König cover C_A of $G[A, B \cup X]$ and any minimum vertex cover C of $G[A \cup X, B]$.*

1. $A \cap C \subseteq A \cap C_A$
2. $B \cap C \supseteq B \cap C_A$.

Proof. To prove this, we will show that it holds for $X = \{x\}$, and then by transitivity of the subset relation and that a König cover is also a minimum vertex cover, it must hold also when X is any subset of $V(G)$.

Let $A' = A + x$ and $B' = B + x$, and let C' be the A-König cover of the graph $G[A, B]$ (be aware that this graph has one less vertex than G). We will break the proof into four parts, namely $A \cap C \subseteq A \cap C'$, $A \cap C' \subseteq A \cap C_A$, $B \cap C_A \subseteq B \cap C'$, and $B \cap C' \subseteq B \cap C$. Again, by transitivity of the subset relation, this will be sufficient for our proof. We now look at each part separately.

$\underline{A \cap C \subseteq A \cap C'}$: Two cases: $|C| = |C'|$ and $|C| > |C'|$. We do the latter first. This means that $C' \cup \{x\}$ must be a minimum vertex cover of $G[A', B]$. Therefore the A'-König cover C^* of $G[A', B']$ must contain $(C' \cup \{x\}) \cap A'$. This means that C^* is a minimum vertex cover of $G[A, B]$, and by C' being the A-König cover of $G[A, B]$, we have from Lemma 2 that $C' \cap A \supseteq C^* \cap A$. And since C^* is a A'-König cover of $G[A', B]$ we have $C' \cap A' \supseteq C \cap A'$ and can conclude that $C' \cap A \supseteq A \cap C$. Now assume that the two vertex covers are of equal size. Clearly $x \notin C$, as then $C - x$ is a smaller vertex cover of $G[A, B]$ than C', so x is not in C. This means that C is a minimum vertex cover of $G[A, B]$, so all vertices in $A \cap C$ must be in C' by Lemma 2.

$\underline{A \cap C' \subseteq A \cap C_A}$: Suppose C' is smaller than C_A. This means $C' + x$ is a minimum vertex cover of $G[A, B']$, and hence $(C' + x) \cap A \subseteq C_A \cap A$ by Lemma 2. On the other hand, if C' is of the same size as C_A. Then C_A is a minimum vertex cover of $G[A, B]$, and so $x \notin C_A$. This means $C_A \cap N(x) \cap A \subseteq C_A \cap A$. And as C_A is a minimum vertex cover of $G[A, B]$, we know from Lemma 2 that $C_A \cap N(x) \cap A \subseteq C'$. In particular, this means C' covers all the edges of $G[A, B']$ not in $G[A, B]$, which means that C' is also a minimum vertex cover of $G[A, B']$. This latter observation means that $C' \cap A \subseteq C_A \cap A$ from Lemma 2.

$\underline{B \cap C_A \subseteq B \cap C'}$: Suppose C' is smaller than C_A. This means $C' + x$ is a minimum vertex cover of $G[A, B']$, and thus $B' \cap (C' + x) \supseteq B' \cap C_A$. Which implies that $B \cap C' \supseteq B \cap C_A$. Now assume that C' is of the same size as C_A. This means C_A is a minimum vertex cover of $G[A, B]$ and $x \notin C_A$. Furthermore, this means $N(x) \cap A \subseteq C_A \cap A \subseteq C' \cap A$ by Lemma 2 and we conclude that C' is a minimum vertex cover of $G[A, B']$. By Lemma 2, this means $B' \cap C_A \subseteq B' \cap C'$ and in particular $B \cap C_A \subseteq B \cap C'$.

$\underline{B \cap C' \supseteq B \cap C}$: Suppose C' is smaller than C. This means $C' + x$ is a minimum vertex cover of $G[A, B']$, and hence by Lemma 2 we have $B' \cap (C' + x) \subseteq B' \cap C_2$, which implies $B \cap C' \subseteq B \cap C_2$. Now suppose C' is of the same size as C. This means that C is a minimum vertex cover of $G[A, B]$, and hence we immediately get $C \cap B \supseteq C' \cap B$ by Lemma 2.

This completes the proof, as we by transitivity of the subset relation have that $C_A \cap B \subseteq C \cap B$, and $C \cap A \subseteq C_A \cap A$. ◀

We are now ready to prove an important connectedness property of König covers that arise from cuts of a given branch decomposition.

▶ **Lemma 4.** *Given a connected graph G and rooted branch decomposition (T, δ) over $V(G)$, for any node v in T, where $\mathcal{C}$ are the descendants of v and C_u means the $\delta(u)$-König cover of $G[\overline{\delta(u)}, \delta(u)]$, we have that*

$$\left(\bigcup_{x \in V(T) \setminus \mathcal{C}} C_x \right) \cap \left(\bigcup_{x \in \mathcal{C}} C_x \right) \subseteq C_v \ .$$

Proof. For all $x \in \mathcal{C}$, since C_x is a $\delta(x)$-König cover and C_v is a minimum vertex cover, from Lemma 3, we have that $C_x \cap \overline{\delta(x)} \subseteq C_v \cap \overline{\delta(x)}$. In particular, since $\delta(x) \subseteq \delta(v)$, we

have that $C_x \setminus \delta(v) \subseteq C_v \setminus \delta(v) \subseteq C_v$. Since each vertex of $V(G)$ is either in $\delta(v)$ or not in $\delta(v)$, by showing that also for all $x \in (V(T) \setminus \mathcal{C})$ we have $C_x \cap \delta(v) \subseteq C_v$ we can conclude that the lemma holds: For all $x \in V(T) \setminus \mathcal{C}$ either $\delta(v) \subseteq \delta(x)$ (when x is an ancestor of v) or $\delta(v) \subseteq \overline{\delta(x)}$ (when x is neither a descendant of v nor an ancestor of v), in either case, we can apply the $\delta(v)$-König cover C_v of $G[\delta(v), \overline{\delta(v)}]$ and the minimum vertex cover C_x of $G[\delta(x), \overline{\delta(x)}]$ to Lemma 3 and see that $C_x \cap \delta(v) \subseteq C_v \cap \delta(v) \subseteq C_v$. ◀

3.2 The new characterization of mmw

We say a graph is *nontrivial* if it has an edge.

▶ **Theorem 5.** *A nontrivial graph $G = (V, E)$ has* $\mathrm{mmw}(G) \leq k$ *if and only if there exist a tree T of max degree at most* 3 *and for each vertex $u \in V$ a nontrivial subtree T_u of T such that* (i) *if $uv \in E$ then the subtrees T_u and T_v have at least one vertex of T in common, and* (ii) *for every edge of T there are at most k subtrees using this edge.*

Proof. Forward direction: Let (T, δ) be a rooted branch decomposition over V having mm-width at most k, and assume G has no isolated vertices. For each edge $e = uv$ of T, with u a child of v, assign the $\delta(u)$-König cover C_u of $G[\delta(u), V \setminus \delta(u)]$ to the edge uv. For each vertex x of G, define the set of edges of T whose König cover contains x and let T_x be the sub-forest of T induced by these edges. Using Lemma 4 we first show that T_x is a connected forest and thus a subtree of T. Consider two vertices u and v such that $x \in C_u \cap C_v$. Let p be the lowest common ancestor of u and v. For every vertex w on the path from p to u and on the path from p to v, except p, we know that exactly one of u, v is a descendant of w. By Lemma 4, $(C_u \cap C_v) \subseteq C_w$. It means that if a vertex x of G is in both C_u and C_v then it is also in C_w, which implies that T_x is connected.

Now, since the branch decomposition has mm-width at most k part (ii) in the statement of the Theorem holds. For an arbitrary edge ab of G, consider any edge e of T on the path from $\delta^{-1}(a)$ to $\delta^{-1}(b)$ and the partition (A, B) induced by e where $a \in A$, $b \in B$. Then the König cover of e must contain one of a and b, and thus, (i) holds as well. Finally, T_x is nontrivial because the edge of T incident with a leaf $\delta^{-1}(x)$ assigns the König cover $\{x\}$. If G has isolated isolated vertices, T_x is not nontrivial for isolated vertex x. We fix this by setting T_x to consist exactly of the edge incident with $\delta^{-1}(x)$, for any isolated vertex x of G.

Backward direction: For each given subtree $\{T_u\}_{u \in V}$ of T, choose an edge in T_u (it is also in T) and append in the tree T a leaf ℓ_u, and extend T_u to contain ℓ_u and set $\delta(\ell_u) = u$. Exhaustively remove leaves (from both T and the subtrees) that are not mapped by δ. Call the resulting tree T' and subtrees $\{T'_u\}_{u \in V}$. Note that subtrees $\{T'_u\}_{u \in V}$ and T' still satisfy (i) and (ii). We claim that (T', δ) is a branch decomposition of mm-width at most k. It is clearly a branch decomposition over V, and for any edge e of T', if we choose $S \subseteq V$ to be those u with T_u using this edge e, then this will be a vertex cover of the bipartite graph H given by this edge e, and of size at most k because for an edge xy in H, one of T_x and T_y must contain e. ◀

In the Introduction we mentioned analogous characterizations of treewidth and branch-width, for these see e.g. [14]. Another alternative characterization is the following.

▶ **Corollary 6.** *A graph G has* $\mathrm{mmw}(G) \leq k$ *if and only if it is a subgraph of a chordal graph H and for every maximal clique X of H there exists $A, B, C \subseteq X$ with $A \cup B \cup C = X$ and $|A|, |B|, |C| \leq k$ such that any subset of X that is a minimal separator of H is a subset of either A, B or C.*

We only sketch the proof, which is similar to an alternative characterization of branchwidth given in [14]. We say a tree is *ternary* if it has maximum degree at most 3. Note that a graph is chordal if and only if it is an intersection graph of subtrees of a tree [8]. In the forward direction, take the chordal graph resulting from the subtrees of ternary tree representation. In the backward direction, take a clique tree of H and make a ternary tree decomposition (which is easily made into a subtrees of ternary tree representation) by for each maximal clique X of degree larger than three making a bag X with three neighboring bags A, B, C. If minimal separators $S_1, ..., S_q \subseteq X$ are contained in A make a path extending from bag A of q new bags also containing A, with a single bag containing $S_i, 1 \leq i \leq q$, attached to each of them. These ternary subtrees, one for each maximal clique, is then connected together in a tree by the structure of the clique tree, adding an edge between bags of identical minimal separators.

4 Fast DP for Dominating Set parameterized by mm-width

For graph $G = (V, E)$ a subset of vertices $S \subseteq V$ is said to *dominate* the vertices in $N[S]$, and it is a *dominating set* if $N[S] = V$. Given a rooted branch decomposition (T, δ) of G of mm-width k, we will in this section give an $O^*(8^k)$ algorithm for computing the size of a Minimum Dominating Set of G. This is an algorithm doing dynamic programming along a rooted tree decomposition $(T', \{X_t\}_{t \in V(T')})$ of G that we compute from (T, δ) as follows.

Given a rooted branch decomposition (T, δ) of G having mm-width k the proof of Theorem 5 yields a polynomial-time algorithm (using an algorithm for maximum matching in bipartite graphs) finding a family $\{T_u\}_{u \in V(G)}$ of nontrivial subtrees of T (note we can assume T is a rooted tree with root of degree two and all other internal vertices of degree three) such that (i) if $uv \in E(G)$ then the subtrees T_u and T_v have at least one vertex of T in common, and (ii) for every edge of T there are at most k subtrees using this edge. From this it is easy to construct a rooted tree decomposition $(T', \{X_t\}_{t \in V(T')})$ of G, having the properties described in Figure 2. Let T' be a tree with vertex set $A \cup B \cup \{r\}$ where A is the set of edges of T, B is the set of non-root vertices (all of degree-3) of T, and r is the root of T and also the root of T'. Two vertices e, v of T' are adjacent if and only if $e \in A$ and $v \in B \cup \{r\}$ are incident in T. For a vertex $e \in A$, let X_e be the set of vertices in G such that if a subtree T_w uses edge e of T, then $w \in X_e$. For a vertex $v \in B$, let X_v be the set of vertices in G such that for the three incident edges e_1, e_2, e_3 of v in T, $X_v = X_{e_1} \cup X_{e_2} \cup X_{e_3}$. Let $X_r = X_{e_1} \cup X_{e_2}$ if e_1 and e_2 are incident with r in T. Then $(T', \{X_t\}_{t \in V(T')})$ is a tree decomposition of G with a root r, having the properties described in Figure 2, which we will use in the dynamic programming.

Let us now define the relevant subproblems for the dynamic programming over this tree decomposition. For node t of the tree we denote by G_t the graph induced by the union of X_u where u is a descendant of t. A coloring of a bag X_t is a mapping $f : X_t \to \{1, 0, *\}$ with the meaning that: all vertices with color 1 are contained in the dominating set of this partial solution in G_t, all vertices with color 0 are dominated, while vertices with color * might be dominated, not dominated, or in the dominating set. Thus the only restriction is that a vertex with color 1 must be a dominator, and a vertex with color 0 must be dominated. Thus, for any $S \subseteq V(G)$ there is a set $c(S)$ of $3^{|S|}2^{|N(S)|}$ colorings $f : V(G) \to \{1, 0, *\}$ compatible with taking S as set of dominators, with vertices of S colored 1, 0 or $*$, vertices of $N(S)$ colored 0 or $*$, and the remaining vertices colored $*$.

For a coloring f of bag X_t we denote by $T[t, f]$ (and view this as a 'Table' of values) the minimum $|S|$ over all $S \subseteq V(G_t)$ such that there exists $f' \in c(S)$ with $f'|_{X_t} = f$ and

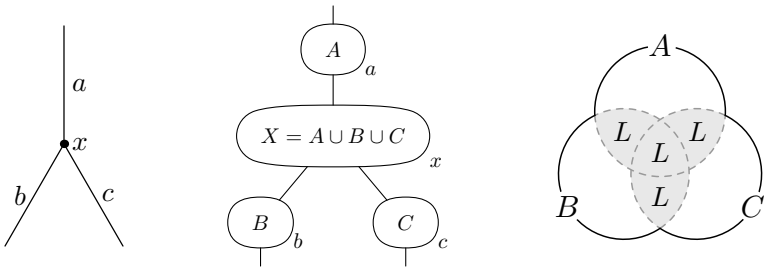


Figure 2 Part of ternary tree used in the subtree representation of G on the left, with node x having three incident edges a, b, c, with subtrees of vertices contained in $A, B, C \subseteq V(G)$ using these edges respectively, giving rise to the four bags in the tree decomposition shown in the middle, with constraint $|A|, |B|, |C| \leq k$.

$f'|_{V(G_t)\setminus X_t}$ having everywhere the value 0. In other words, the minimum size of a set S of vertices of G_t that dominate all vertices in $V(G_t) \setminus X_t$, with a coloring f' compatible with taking S as set of dominators, such that f' restricted to X_t gives f. If no such set S exists, then $T[t, f] = \infty$. Note that the size of the minimum dominating set of G is the minimum value over all $T[r, f]$ where $f^{-1}(*) = \emptyset$ at the root r. We initialize the table at a leaf ℓ, with $X_\ell = \{v\}$ as follows. Denote by f_i the coloring from $\{v\}$ to $\{1, 0, *\}$ with $f_i(v) = i$ for $i \in \{1, 0, *\}$. Then for a leaf bag X_ℓ, set $T[\ell, f_1] := 1$, $T[\ell, f_0] := \infty$, $T[\ell, f_*] := 0$.

For internal nodes of the tree, instead of separate 'Join, Introduce and Forget' operations we will give a single update rule with several stages. We will be using an Extend-Table subroutine which takes a partially filled table $T[t, \cdot]$ and extends it to table $T'[t, \cdot]$ so the result will adhere to the above definition, ensuring the monotonicity property that $T'[t, f] \leq T'[t, f']$ for any f we can get from f' by changing the color of a vertex from 1 to 0 or $*$, or from 0 to $*$. Extend-Table is implemented as follows:

(a) Initialize. For all f, if $T[t, f]$ is defined then $T'[t, f] := T[t, f]$, else $T'[t, f] := \infty$.

(b) Change from 1 to 0. For $q = |X_t|$ down to 1: for any f in $T'[t, f]$ where $|\{v : f(v) = 1\}| = q$, for any choice of a single vertex $u \in \{v : f(v) = 1\}$ set $f_u(u) = 0$ and set $f_u(x) = f(x)$ for $x \neq u$, and update $T'[t, f_u] := \min\{T'[t, f_u], T'[t, f]\}$.

(c) Change from 0 to $*$. Similarly as in step (b).

Note the transition from color 1 to $*$ will happen by transitivity. The time for Extend-Table is proportional to the number of entries in the tables times $|X_t|$.

Assume we have the situation in Figure 2, corresponding to the bags surrounding any degree-three node x of the tree decomposition. This arises from the branch decomposition (and the subtrees of tree representation) having a node incident to three edges, creating three bags a, b, c containing subsets of vertices A, B, C, respectively, each of size at most k, and giving rise to the four bags a, b, c, x in Figure 2, with the latter containing subsets of vertices $X = A \cup B \cup C$. Let $L = (A \cap B) \cup (A \cap C) \cup (B \cap C)$. Assume we have already computed $T[b, f]$ and $T[c, f]$ for all $3^{|B|}$ and $3^{|C|}$ choices of f, respectively. We want to compute $T[a, f]$ for all $3^{|A|}$ choices of f, in time $O^*(\max\{3^{|A|}, 3^{|B|}, 3^{|C|}, 3^{|L|}2^{|X\setminus L|}\})$. Note that we will not compute the table $T[x, \cdot]$, as it would have $3^{|X|}$ entries, which is more than the allowed time bound. Instead, we compute a series of tables:

(1) $T_b^1[x, \cdot]$ (and $T_c^1[x, \cdot]$) of size $3^{|B|}$, by for each entry $T[b, f]$ extending the coloring f of B to a unique coloring f' of X based on the neighborhood of the dominators in f

(2) $T_b^2[x, \cdot]$ (and $T_c^2[x, \cdot]$) of size at most $\min(3^{|B|}, 3^{|B\cap L|}2^{|X\setminus(B\cap L)|})$, by changing each coloring f of X to a coloring f' of X where vertices in $B \setminus L$ having color 1 instead are given color 0 (note these vertices have no neighbors in $V(G) \setminus V(G_x)$)

(3) $T_b^3[x,\cdot]$ (and $T_c^3[x,\cdot]$) of size exactly $3^{|B\cap L|}2^{|X\setminus(B\cap L)|}$, with $f^{-1}(1)\subseteq B\cap L$, by running Extend-Table on $T_b^2[x,\cdot]$
(4) $T_{sc}^1[x,\cdot]$ of size $3^{|L|}2^{|X\setminus L|}$ by subset convolution over parts of $T_b^3[x,\cdot]$ and $T_c^3[x,\cdot]$
(5) $T_{sc}^2[x,\cdot]$ of size $3^{|L|}2^{|X\setminus L|}$ by running Extend-Table on $T_{sc}^1[x,\cdot]$
(6) $T[a,\cdot]$ of size $3^{|A|}$ by going over all $3^{|A|}$ colorings of A and minimizing over appropriate entries of $T_{sc}^2[x,\cdot]$

Note that in step 4 we use the following:

▶ **Theorem 7** (Fast Subset Convolution [2]). *For two functions $g,h:2^V\to\{-M,\ldots,M\}$, given all the $2^{|V|}$ values of g and h in the input, all $2^{|V|}$ values of the subset convolution of g and h over the integer min-sum semiring, i.e. $(g*h)(Y)=\min_{Q\cup R=Y \text{ and } Q\cap R=\emptyset} g(Q)+h(R)$, can be computed in time $2^{|V|}|V|^{O(1)}\cdot O(M\log M\log\log M)$.*

Let us now give the details of the first three steps:

(1) Compute $T_b^1[x,\cdot]$. In any order, go through all $f:B\to\{1,0,*\}$ and compute $f':B\cup A\cup C\to\{1,0,*\}$ by

$$f'(v)=\begin{cases} f(x) & \text{if } v\in B\\ 0 & \text{if } v\notin B \text{ and } \exists u\in B: f(u)=1\wedge uv\in E(G)\\ * & \text{otherwise}\end{cases}$$

and set $T_b^1[x,f']:=T[b,f]$.

(2) Compute $T_b^2[x,\cdot]$. First, initialize $T_b^2[x,f]=\infty$ for all $f:B\cup A\cup C\to\{1,0,*\}$ where $f^{-1}(1)\subseteq B\cap L$. In any order, go through all $f:B\cup A\cup C\to\{1,0,*\}$ such that $T_b^1[x,f]$ was defined in the previous step, and compute $f':B\cup A\cup C\to\{1,0,*\}$ by

$$f'(v)=\begin{cases} 0 & \text{if } v\in B\setminus L \text{ and } f(v)=1\\ f'(v)=f(v) & \text{otherwise}\end{cases}$$

and set $T_b^2[x,f']:=\min\{T_b^2[x,f'],T_b^1[x,f]\}$. There will be no other entries in $T_b^2[x,\cdot]$.

(3) Compute $T_b^3[x,\cdot]$ by Extend-Table on $T_b^2[x,\cdot]$.

The total time for the above three steps is bounded by $O^*(\max\{3^{|B|},3^{|B\cap L|}2^{|X\setminus(B\cap L)|}\})$. Note that $T_b^3[x,f]$ is defined for all f where vertices in $B\cap L$ take on values $\{1,0,*\}$ and vertices in $X\setminus(B\cap L)$ take on values $\{0,*\}$. The value of $T_b^3[x,f]$ will be the minimum $|S|$ over all $S\subseteq V(G_b)$ such that there exists $f'\in c(S)$ with $f'|_X=f$ and $f'|_{V(G_b)\setminus X}$ having everywhere the value 0. Note the slight difference from the standard definition, namely that even though the coloring f is defined on X, the dominators only come from $V(G_b)$, and not from $V(G_x)$. The table $T_c^3[x,\cdot]$ is computed in a similar way, with the colorings again defined on X but with the dominators now coming from $V(G_c)$.

When computing a Join of these two tables, we want dominators to come from $V(G_b)\cup V(G_c)$. Because of the monotonicity property that holds for these two tables, we can compute their Join $T_{sc}^1[x,f]$ for any f where vertices in L take on values $\{1,0,*\}$ and vertices in $X\setminus L$ take on values $\{0,*\}$, by combining colorings as follows:

$$T_{sc}^1[x,f]=\min_{f_b,f_c}(T_b^3[x,f_b]+T_c^3[x,f_c])-|f^{-1}(1)\cap B\cap C|$$

where f_b,f_c satisfy:

- $f(v)=0$ if and only if $(f_b(v),f_c(v))\in\{(0,*),(*,0)\}$
- $f(v)=*$ if and only if $f_b(v)=f_c(v)=*$
- $f(v)=1$ if and only if $v\in B\cap C$ and $f_b(v)=f_c(v)=1$, or $v\in B\setminus C$ and $(f_b(v),f_c(v))=(1,*)$, or $v\in C\setminus B$ and $(f_b(v),f_c(v))=(*,1)$.

This means that we can apply subset convolution to compute a table $T^1_{sc}[x, f]$ on $3^{|L|}2^{|X\setminus L|}$ entries based on $T^3_b[x, f]$ and $T^3_c[x, f]$. Note that $(B \cap L) \cup (C \cap L) = L$. For this step we follow the description in [6, Section 11.1.2]. Fix a set $D \subseteq L$ to be the dominating vertices. Let F_D denote the set of $2^{|X\setminus D|}$ functions $f : X \to \{1, 0, *\}$ such that $f^{-1}(1) = D$, i.e. with vertices in $X \setminus D$ mapping in all possible ways to $\{0, *\}$. For each $D \subseteq L$ we will by subset convolution compute the values of $T^1_{sc}[x, f]$ for all $f \in F_D$.

We represent every $f \in F_D$ by the set $S = f^{-1}(0)$ and define $b_S : X \to \{1, 0, *\}$ such that $b_S(x) = 1$ if $x \in D \cap B$, $b_S(x) = 0$ if $x \in S$, $b_S(x) = *$ otherwise. Similarly, define $c_S : X \to \{1, 0, *\}$ such that $c_S(x) = 1$ if $x \in D \cap C$, $c_S(x) = 0$ if $x \in S$, $c_S(x) = *$ otherwise. Then, as explained previously, for every $f \in F_D$ we want to compute

$$T^1_{sc}[x, f] = \min_{Q\cup R=f^{-1}(0) \text{ and } Q\cap R=\emptyset} (T^3_b[x, b_Q] + T^3_c[x, c_R]) - |f^{-1}(1) \cap B \cap C|.$$

Define functions $T_b : 2^{X\setminus D} \to \mathbb{N}$ such that for every $S \subseteq X \setminus D$ we have $T_b(S) = T^3_b[x, b_S]$. Likewise, define functions $T_c : 2^{X\setminus D} \to \mathbb{N}$ such that for every $S \subseteq X \setminus D$ we have $T_c(S) = T^3_c[x, c_S]$. Also, define $a_S : X \to \{1, 0, *\}$ such that $a_S(x) = 1$ if $x \in D$, $a_S(x) = 0$ if $x \in S$, $a_S(x) = *$ otherwise. We then compute for every $S \subseteq X \setminus D$,

$$T^1_{sc}[x, a_S] := (T_b * T_c)(S) - |f^{-1}(1) \cap B \cap C|$$

where the subset convolution is over the mini-sum semiring.

(4) In step (4), by Fast Subset Convolution, Theorem 7, we compute $T^1_{sc}[x, a_S]$, for all a_S defined by all $f \in F_D$, in $O^*(2^{|X\setminus D|})$ time each. For all such subsets $D \subseteq L$ we get the time

$$\sum_{D\subseteq L} 2^{|X\setminus D|} = \sum_{D\subseteq L} 2^{|X\setminus L|}2^{|L\setminus D|} = 2^{|X\setminus L|} \sum_{D\subseteq L} 2^{|L\setminus D|} = 2^{|X\setminus L|}3^{|L|}.$$

(5) In step (5) we need to run Extend-Table on $T^1_{sc}[x, \cdot]$ to get the table $T^2_{sc}[x, \cdot]$. This since the subset convolution was computed for each fixed set of dominators so the monotonicity property of the table may not hold. Note that the value of $T^2_{sc}[x, f]$ will be the minimum $|S|$ over all $S \subseteq V(G_b) \cup V(G_c)$ such that there exists $f' \in c(S)$ with $f'|_X = f$ and $f'|_{(V(G_b)\cup V(G_c))\setminus X}$ having everywhere the value 0.

(6) In step (6) we will for each $f : A \to \{1, 0, *\}$ compute $f' : B \cup A \cup C \to \{1, 0, *\}$ by

$$f'(v) = \begin{cases} 1 & \text{if } v \in A \cap L \text{ and } f(v) = 1 \\ 0 & \text{if } v \in A \text{ and } f(v) = 0 \text{ and } N(v) \cap f^{-1}(1) = \emptyset \\ 0 & \text{if } v \notin A \text{ and } N(v) \cap f^{-1}(1) = \emptyset \\ * & \text{otherwise} \end{cases}$$

and set $T[a, f] := T^2_{sc}[x, f'] + |f^{-1}(1) \cap (A \setminus L)|$.

Note that when we iterate over all choices of $f : A \to \{1, 0, *\}$, the vertices colored 0 (in addition to all vertices of $X \setminus A$) must either be dominated by the vertices in $f^{-1}(1)$ or by vertices in $X \setminus V_a$. As we know precisely what vertices of $f^{-1}(0)$ are dominated by $f^{-1}(1)$, we know the rest must be dominated from vertices of $X \setminus V_a$, and therefore we look in $T_{sc}[x, f']$ at an index f' which colors the rest of $f^{-1}(0)$ by 0. We can also observe that it is not important for us whether or not $f^{-1}(0)$ contains all neighbours of $f^{-1}(1)$, since we are iterating over all choices of f - also those where $f^{-1}(0)$ contains all neighbours of $f^{-1}(1)$.

The total runtime becomes $O^*(\max\{3^{|A|}, 3^{|B|}, 3^{|C|}, 3^{|L|}2^{|(A\cup B\cup C)\setminus L|}\})$, with $L = (A \cap B) \cup (A \cap C) \cup (B \cap C)$ and with constraints $|A|, |B|, |C| \leq k$. This runtime is maximum when $L = \emptyset$, giving a runtime of $O^*(2^{3k})$. We thus have the following theorem.

▶ **Theorem 8.** *Given a graph G and branch decomposition over its vertex set of mm-width k we can solve Dominating Set in time $O^*(8^k)$.*

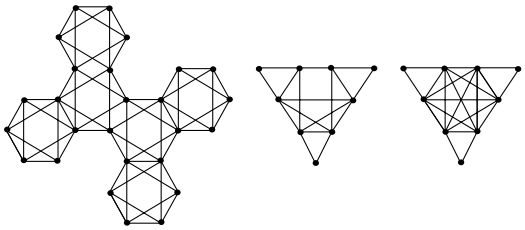


Figure 3 Three graphs of mm-width 2. Left, middle have treewidth 4, and right has treewidth 5.

5 Discussion

We have shown that the graph parameter mm-width will for some graphs be better than treewidth for solving Minimum Dominating Set. The improvement holds whenever $\mathrm{tw}(G) > 1.549 \times \mathrm{mmw}(G)$, if given only the graph as input. In Figure 3 we list some examples of small graphs having treewidth at least twice as big as mm-width. It could be interesting to explore the relation between treewidth and mm-width for various well-known classes of graphs. The given algorithmic technique, using fast subset convolution, should extend to any graph problem expressible as a maximization or minimization over (σ, ρ)-sets, using the techniques introduced for treewidth in [17].

We may also compare with branchwidth. Let ω be the *exponent of matrix multiplication*, which is less than 2.3728639 [11]. In 2010, Bodlaender, van Leeuwen, van Rooij, and Vatshelle [5] gave an $O^*(3^{\frac{\omega}{2}k})$ time algorithm solving Minimum Dominating Set if an input graph is given with its branch decomposition of width k. This means that given decompositions of $\mathrm{bw}(G)$ and $\mathrm{mmw}(G)$ our algorithm based on mm-width is faster than the algorithm in [5] whenever $\mathrm{bw}(G) > \log_3 8 \cdot \frac{2}{\omega} \cdot \mathrm{mmw}(G) > \frac{2 \log_3 8}{2.3728639} \cdot \mathrm{mmw}(G) > 1.6\,\mathrm{mmw}(G)$.

Taking the subtrees of tree representation for treewidth, branchwidth and maximum matching width mentioned in the Introduction as input, our algorithm for dominating set can be seen as a generic one that works for any of treewidth, branchwidth or maximum matching width of the given representation, and in case of both treewidth and mm-width it will give the best runtime known.

We gave an alternative definition of mm-width using subtrees of a tree, similar to alternative definitions of treewidth and branchwidth. We saw that in the subtrees of a tree representation treewidth focuses on nodes, branchwidth focuses on edges, and mm-width combines them both. There is also a fourth way of defining a parameter through these intersections of subtrees representation; where subtrees T_u and T_v must share an edge if $uv \in E(G)$ (similar to branchwidth) and the width is defined by the maximum number of subtrees sharing a single vertex (similar to treewidth). This parameter will be an upper bound on all the other three parameters, but might it be that the structure this parameter highlights can be used to improve the runtime of Dominating Set beyond $O^*(3^{\mathrm{tw}(G)})$ for even more cases than those shown using mm-width and branchwidth?

References

1 Eyal Amir. Approximation algorithms for treewidth. *Algorithmica*, 56(4):448–479, 2010.

2 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets Möbius: fast subset convolution. In *STOC'07 – Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, pages 67–74. ACM, New York, 2007.

3 Hans L Bodlaender, Pal Gronas Drange, Markus S Dregi, Fedor V Fomin, Daniel Lokshtanov, and Michal Pilipczuk. An $o(c^k n)$ 5-approximation algorithm for treewidth. In

Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on, pages 499–508. IEEE, 2013.

4 Hans L. Bodlaender and Ton Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *J. Algorithms*, 21(2):358–402, 1996.

5 Hans L. Bodlaender, Erik Jan van Leeuwen, Johan M. M. van Rooij, and Martin Vatshelle. Faster algorithms on branch and clique decompositions. In *Mathematical foundations of computer science 2010*, volume 6281 of *Lecture Notes in Comput. Sci.*, pages 174–185. Springer, Berlin, 2010.

6 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer International Publishing, New York, 2016.

7 Reinhard Diestel. *Graph theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, Heidelberg, fourth edition, 2010.

8 Fănică Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *J. Combinatorial Theory Ser. B*, 16:47–56, 1974.

9 Jisu Jeong, Sigve Hortemo Sæther, and Jan Arne Telle. An FPT algorithm computing a decomposition of optimal mm-width. in preparation, 2015.

10 Dénes König. Gráfok és mátrixok. *Matematikai és Fizikai Lapok*, 38:116–119, 1931.

11 François Le Gall. Powers of tensors and fast matrix multiplication. In *ISSAC 2014 – Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, pages 296–303. ACM, New York, 2014.

12 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, Philadelphia, PA, 2011.

13 Sang-il Oum and Paul Seymour. Approximating clique-width and branch-width. *J. Combin. Theory Ser. B*, 96(4):514–528, 2006.

14 Christophe Paul and Jan Arne Telle. Edge-maximal graphs of branchwidth k: the k-branches. *Discrete Math.*, 309(6):1467–1475, 2009.

15 Neil Robertson and P. D. Seymour. Graph minors. X. Obstructions to tree-decomposition. *J. Combin. Theory Ser. B*, 52(2):153–190, 1991.

16 Sigve Hortemo Sæther and Jan Arne Telle. Between treewidth and clique-width. In *Graph-theoretic concepts in computer science*, volume 8747 of *Lecture Notes in Comput. Sci.*, pages 396–407. Springer, Cham, 2014.

17 Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In *Algorithms – ESA 2009*, volume 5757 of *Lecture Notes in Comput. Sci.*, pages 566–577. Springer, Berlin, 2009.

18 Martin Vatshelle. *New Width Parameters of Graphs*. PhD thesis, University of Bergen, 2012.

Fast Parallel Fixed-parameter Algorithms via Color Coding*

Max Bannach[1,2], Christoph Stockhusen[1], and Till Tantau[1]

1 **Institute for Theoretical Computer Science**
Universität zu Lübeck
Lübeck, Germany
{bannach,stockhus,tantau}@tcs.uni-luebeck.de
2 **Graduate School for Computing in Medicine and Life Sciences**
Universität zu Lübeck, Germany

Abstract

Fixed-parameter algorithms have been successfully applied to solve numerous difficult problems within acceptable time bounds on large inputs. However, most fixed-parameter algorithms are inherently *sequential* and, thus, make no use of the parallel hardware present in modern computers. We show that parallel fixed-parameter algorithms do not only exist for numerous parameterized problems from the literature – including vertex cover, packing problems, cluster editing, cutting vertices, finding embeddings, or finding matchings – but that there are parallel algorithms working in *constant* time or at least in time *depending only on the parameter* (and not on the size of the input) for these problems. Phrased in terms of complexity classes, we place numerous natural parameterized problems in parameterized versions of AC^0. On a more technical level, we show how the *color coding* method can be implemented in constant time and apply it to embedding problems for graphs of bounded tree-width or tree-depth and to model checking first-order formulas in graphs of bounded degree.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases color coding, parallel computation, fixed-parameter tractability, graph packing, cutting ℓ vertices, cluster editing, tree-width, tree-depth, model checking

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.224

1 Introduction

The classical objective of parameterized complexity theory is to determine for a parameterized problem whether it can be solved by an algorithm running in time $f(k) \cdot n^c$, where f is some function, k is a parameter, n is the input length, and c is some constant. Such algorithms are nowadays routinely used to solve large instances for NP- or even PSPACE-hard problems within acceptable amounts of time. Nevertheless, "acceptable" is not the same as "small" and one would like to further reduce the runtime by using multiple cores to speed up the computation. For this, one needs *parallel* fixed-parameter algorithms, but most fixed-parameter algorithms have been devised with a sequential computation model in mind. Indeed, the most important tool of parameterized complexity theory, namely kernelization, is inherently sequential: It asks us to repeatedly apply rules to an input, each time modifying the input slightly and making it a little smaller, until the input's size only depends on the parameter. There is no straightforward way of parallelizing such algorithms since later

* A full version of this paper is available under http://arxiv.org/abs/1509.06984.

10th International Symposium on Parameterized and Exact Computation (IPEC 2015).
Editors: Thore Husfeldt and Iyad Kanj; pp. 224–235

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

modifications strongly depend on what happened earlier, forcing us to apply the typically very large number of kernelization steps in a sequential manner.

Our Contributions. The purpose of the present paper is to show that not only do parallel fixed-parameter algorithms exist for many natural, well-studied problems from the literature; for certain problems there are even parallel algorithms that require only *constant* time in a concurrent-read, concurrent-write PRAM model (so the runtime is totally independent of the input) or at least time *depending only on the parameter* (so the length of the input is irrelevant). In all cases, the *work* done by the algorithms is still $f(k) \cdot n^c$, that is, the same as the time bound for sequential fixed-parameter algorithms.[1] Phrased more formally, our objective is to identify parameterized problems that lie in the complexity classes para-$\mathrm{AC}_{O(1)}$ and para-$\mathrm{AC}_{f(k)}$ (formal definitions will be given later).

In order to tackle the parallel parameterized complexity of natural problems like the vertex cover problem, we introduce three technical tools. The first and foremost is *color coding*: all of our proofs employ this technique at least indirectly and we show that the *universal coloring families* that lie at the heart of the technique can be computed in constant time. Second, numerous natural "packing problems" are special cases of the following embedding problem: Given graphs H and G, find a (not necessarily induced) subgraph of G that is isomorphic to H. We give new bounds on the complexity of this problem when H has bounded tree-width or bounded tree-depth; and these bounds later translate directly to bounds on different packing problems. Third, we translate an algorithmic meta-theorem of Flum and Grohe [16] to the parallel world: We show that model checking first-order properties of graphs can be done in parallel in time depending only on the parameters (actually, only on the locality rank of the formula), where the parameters are the to-be-checked formula and the degree of the graph.

We then apply the tools to a wide variety of natural graph problems, namely *packing problems, covering problems, clustering problems,* and *separation problems.* For *packing problems* the objective is to determine whether a given graph G contains k vertex-disjoint copies of some fixed graph H like, say, a triangle. Even for triangles, this problem is already NP-complete, but when k is considered as a parameter, the triangle packing problem lies in FPT [15]. We show that there is a constant-time, FPT-work algorithm for triangle packing – and indeed for packing any graph of fixed size. The *covering problems* we study include the vertex cover problem and its partial version. We present a constant parallel time algorithm for the first problem and an algorithm for the second needing time depending only on the parameter. These results nicely reflect on a theoretical basis the "empirical" observation that p-VERTEX-COVER is one of the "easiest" parameterized problems and that the partial version is a bit harder to solve. For *clustering problems,* also known as *cluster editing problems,* the objective is to transform a graph by adding or deleting few edges into a collection of "clusters" – which are just cliques in the simplest case. We present a constant time, FPT-work algorithm for cluster editing. For *graph separation problems* the objective is to "cut away" a special part of a graph using few vertices. We show that certain versions of these problems can be solved by a parallel fixed-parameter algorithm in time depending only on the parameter and FPT work (while other versions are known to be W[1]-hard).

[1] The *work* done by a parallel algorithm is the total number of computational steps made by all computational units during a computation. Since "all work needs to be done," in practice the runtime of a parallel algorithm is its work divided by the number of available cores. In particular, the *work* done by a *parallel algorithm* should not exceed the *runtime* of a *sequential algorithm* for the same problem. In our case, this means that in order to compete with sequential algorithms running in "FPT time," our parallel algorithm must not only be fast, but may only do "FPT work."

Related Work. There is a growing body of literature reporting on the practicalities of implementing fixed-parameter algorithms in parallel [1]. In contrast, there are only few results addressing parallel fixed-parameter tractability on a theoretical level (as we do in the present paper), see for instance Cesati and Di Ianni [9]. Since it is well-known from classical complexity theory that problems solvable in logarithmic space can be parallelized well, previous research on *parameterized logarithmic space* contributes to our understanding of which parameterized problems can be parallelized in principle. This research was started by Cai, Chen, Downey, and Fellows [8]. First (quite technical) complete problems for parameterized logarithmic space where later introduced by Chen, Flum, and Grohe [11], and by Flum and Grohe [16]. A more structural study of parameterized space and circuit classes (which addresses parallelization more directly) was later made by Elberfeld and the last two authors [14]. Parameterized Circuit Complexity was also studied by Downey et al. with respect to the Weft Hierarchy [13]. Recently, Chen and Müller [10] connected color coding and parameterized space in an algorithm for finding embedding of bounded tree-depth graphs in parameterized logarithmic space (a result which we strengthen considerably in Corollary 3.7).

The first use of the color coding technique can be traced back to Alon, Yuster, and Zwick [2]. They used the technique to provide an FPT-algorithm that decides whether there is an embedding of a graph H of bounded tree-*width* into another graph G, where H is the parameter.

Organization of This Paper. In Section 2 we give formal definitions of the classes of problems solvable by parallel fixed-parameter algorithms. While most of our definitions and classes are standard, the class of problems solvable in "time depending on the parameter and FPT work" seems to be new. In Section 3 we introduce our three technical tools – color coding, embeddings, and model checking – and prove the results mentioned earlier. In Section 4 we study the complexity of the natural parameterized graph problems and establish new upper bounds on their complexities. Due to lack of space, most proofs are only available in the full version; we give proof sketches for some of them in the main text.

2 Classes of Fixed-parameter Parallelism

For our definition of parallel fixed-parameter tractability, we mostly use the standard terminology of parameterized complexity theory, see for instance [17]: A *parameterized problem* is a tuple (Q, κ) of a language $Q \subseteq \Sigma^*$ over an alphabet Σ and a parameterization $\kappa\colon \Sigma^* \to \mathbb{N}$ that maps instances to parameter values. In the classical definition, Downey and Fellows [12] require the parameterization to be computable, while Flum and Grohe [17] require it to be computable in polynomial time. Elberfeld and the last two authors require it to be computable in logarithmic space [14] and mention that it would be better if the parameterization is first-order computable (FO-computable) or, equivalently, to be computable by logarithmic-time-uniform constant depth circuits [24]. Since we will only deal with parameterized circuit classes that lie within parameterized logarithmic space, we will require all parameterizations to be FO-computable. We denote parameterized problems with a leading "p-" as in p-VERTEX-COVER and, when the parameter may be unclear, add it as an index as in $p_{|H|}$-EMB.

A parameterized problem is *fixed-parameter tractable* if it can be decided in time $f(\kappa(x)) \cdot |x|^c$ for any input x, where f is some computable function and c a constant. An equivalent definition is that there exists a set $R \in \mathrm{P}$, where P denotes the class of languages decidable in

polynomial time, such that $x \in Q$ iff $\left(x, 1^{f(\kappa(x))}\right) \in R$. The first definition of fixed-parameter tractability gave rise to the class name FPT in the literature, while the second definition gives rise to the name para-P for the same class. The advantage of the second definition is that we can replace the class P in the definition by arbitrary complexity classes and arrive at classes like *parameterized logarithmic space,* para-L, or *parameterized constant depth circuits*, para-AC^0. These parameterized classes inherit their inclusion structure from the classical classes, so we have

$$\text{para-AC}^0 \subsetneq \text{para-TC}^0 \subseteq \text{para-NC}^1 \subseteq \text{para-L} \subseteq \text{para-NL} \subseteq \text{para-AC}^1 \subseteq \text{para-P}.$$

It is not quite obvious, but the class para-AC^0 already captures one of the types of algorithms mentioned in the introduction, namely "constant time, FPT-work," while none of the above classes seems to capture "parameter time, FPT-work." For this reason and in order to explicitly spell out what para-AC^0 contains, we provide a new definition:[2]

▶ **Definition 2.1** (Classes of Parallel Fixed-Parameter Tractability)**.** Let $d\colon \mathbb{N}^2 \to \mathbb{N}$ be a *depth bounding function* and $w\colon \mathbb{N}^2 \to \mathbb{N}$ be a *width bounding function* which both map each pair of an input length and a parameter to a number. We define para-$\mathrm{AC}[d, w]$ as the class of parameterized problems (Q, κ) for which there exists a DLOGTIME-uniform[3] family $(C_{n,k})_{n,k\in\mathbb{N}}$ of AC-circuits (only NOT-, AND-, and OR-gates are allowed, AND- and OR- gates may have unbounded fan-in) such that:

1. For all $x \in \Sigma^*$, the circuit $C_{|x|,\kappa(x)}$ evaluates to 1 on input x if, and only if, $x \in Q$.
2. The depth of each $C_{n,k}$ is at most $d(n, k)$.
3. The size of each $C_{n,k}$ is at most $w(n, k)$.

In the present paper we exclusively study parallel algorithms with "FPT-work" and are therefore only interested in the case where w is member of the family W of functions of the form $f(k) \cdot n^c$ for a computable function f and a constant c. We introduce for arbitrary families D of functions $d\colon \mathbb{N}^2 \to \mathbb{N}$ the abbreviation para-AC_D for $\bigcup_{d\in D, w\in W}$ para-$\mathrm{AC}[d, w]$. For constant depth bounding functions the resulting class para-$\mathrm{AC}_{O(1)}$ is the same as the class para-AC^0.[4] For arbitrary $i > 0$ we obtain para-$\mathrm{AC}_{\{\, f(k)+c\cdot\log^i n \mid f\colon \mathbb{N}\to\mathbb{N}\wedge c\in\mathbb{N}\,\}}$ = para-AC^i(in slight abuse of notation we will write such classes simply as para-$\mathrm{AC}_{f(k)+O(\log^i n)}$).

When the depth bounding function just depends on the parameter, so $d(n, k) = f(k)$, we get a new class para-$\mathrm{AC}_{f(k)}$ that we abbreviate with para-$\mathrm{AC}^{0\uparrow}$. This class does not seem to arise from substituting some classical class for P in the definition of para-P. In particular, this class seems to be incomparable with all classes between para-TC^0 and para-NL. It is, however, clearly contained in para-AC^1, and is strictly more powerful then para-AC^0 as we will see later. This class captures the problems solvable in "parameter time, FPT-work" and we have

$$\text{para-AC}^0 \subsetneq \text{para-AC}^{0\uparrow} \subseteq \text{para-AC}^1.$$

Let us define for arbitrary $i \geq 0$ the class para-$\mathrm{AC}^{i\uparrow}$ as para-$\mathrm{AC}_{f(k)\cdot O(\log^i n)}$. Notice that we have by definition the inclusion structure para-$\mathrm{AC}^i \subseteq$ para-$\mathrm{AC}^{i\uparrow} \subseteq$ para-$\mathrm{AC}^{i+\epsilon}$.

---

[2] The definition can trivially be adjusted to use TC-circuits or NC-circuits, but we will not need them.

[3] We use DLOGTIME-uniform families since they are equivalent to first-order definable families and constitute one of the strongest forms of uniformity [4].

[4] Since the designation para-AC^0 has been used in previous publications and is a bit shorter, we will use it in the following.

3 Technical Tools

3.1 Color Coding in Constant Parallel Time

The idea of color coding is best understood by a concrete application, for instance to the well-known matching problem: Given an undirected graph G and a number k, does G contain k edges such that no two of them share any endpoints? Directly solving this problem is not easy since the known polynomial-time algorithms for it are rather involved. Consider, however, what happens when we randomly color the graph with k colors and then check whether *the vertices of each color class contain at least one edge*. Clearly, if this is the case, there is a matching of size k – and if there is no such matching, then no coloring will pass the test.

We now formalize the idea behind color coding and then show how the colorings can be computed in constant time. It turns out that one can derandomize the computation of a coloring: instead of random colorings we use sets of colorings such that for every set of k vertices and "desired" colors for them, at least one coloring colors the vertices as desired:

▶ **Definition 3.1** (Universal Coloring Families)**.** For natural numbers n, k, and c, an (n, k, c)-*universal coloring family* is a set Λ of functions $\lambda\colon \{1, \dots, n\} \to \{1, \dots, c\}$ such that for every subset $S \subseteq \{1, \dots, n\}$ of size $|S| = k$ and for every mapping $\mu\colon S \to \{1, \dots, c\}$ there is at least one function $\lambda \in \Lambda$ with $\forall s \in S\colon \mu(s) = \lambda(s)$.

The matching problem can be solved easily when we have access to a $(n, 2k, k)$-universal coloring family: If there is a matching of size k, the family will contain some coloring that colors the two endpoints of the first edge with color 1, the endpoints of the second edge with color 2, and so on. Thus there is, indeed, a matching of size k in the graph if for at least one coloring every color class contains an edge. Since we can easily check in parallel for all colorings whether this is the case for one of them, the complexity of p_k-MATCHING hinges critically on the complexity of computing the universal coloring family and the size of this family. The next theorem shows that (n, k, c)-universal coloring families of reasonable size can be computed "in constant time and work $f(k, c) \cdot n^{O(1)}$," which implies that p_k-MATCHING $\in$ para-AC^0 holds:

▶ **Theorem 3.2.** *There is a* DLOGTIME*-uniform family* $(C_{n,k,c})_{n,k,c\in\mathbb{N}}$ *of* AC*-circuits without inputs such that each* $C_{n,k,c}$

1. *outputs an* (n, k, c)*-universal coloring family (coded as a sequence of function tables),*
2. *has constant depth (independent of* n*,* k*, or* c*), and*
3. *has size at most* $O(n \log c \cdot c^{k^2} \cdot k^4 \log^2 n)$*.*

Sketch of Proof. The family of universal coloring functions we construct is based on the concept of k-perfect hash functions [17], that, after slight modifications, provide us with the desired coloring properties. The crucial part is to implement them using circuits that are DLOGTIME-uniform. However, we can achieve this, since the numbers n, k, and c are encoded in unary and the operations required to compute the functions are only additions, multiplications, and modulo operations. ◀

Investigating a parameterized version of matching may seem a bit strange at first sight – matching is even known to be solvable in randomized polylogarithmic parallel time. However, the exact parallel time complexity is still open in the classical setting while from a parameterized perspective, we just saw that the matching can be solved *very* quickly in parallel. Another problem that one would maybe not expect to be studied in the parameterized

setting, but which will be useful in a number of situations, is p-THRESHOLD. The inputs are a bitstring $b \in \{0,1\}^n$ and a parameter t. The question is whether there are at least t many 1's in b. Clearly, the unparameterized version is complete for TC^0, and using the fact that the problem lies in AC^0 for polylogarithmic thresholds [23] yields the fact that its parameterized version lies in para-AC^0. However, this result requires profound result of circuit complexity and is rather involved, but using color coding we can give a very simple proof of this fact:

▶ **Lemma 3.3.** *p-THRESHOLD $\in$ para-AC^0.*

3.2 Finding Embeddings of Graphs of Bounded Tree-Width and Depth

A different way of looking at the matching problem is to see it as an embedding problem: Instead of trying to find k edges in a graph G that have no endpoints in common, we can try to "embed" the graph $H = kK_2$, consisting of k isolated edges, into G. The advantage of this different point of view is, of course, that it generalizes nicely:

▶ **Problem 3.4** (p-EMB($\mathcal{H}$) for some class $\mathcal{H}$ of undirected graphs).

Instance: *Two undirected graphs $H = (V_H, E_H) \in \mathcal{H}$ and $G = (V_G, E_G)$.*
Parameter: *H*
Question: *Is there a injective homomorphism $\phi\colon V_H \to V_G$, that is, is H isomorphic to a (not necessarily induced) subgraph of G?*

For arbitrary $\mathcal{H}$, the problem is easily be seen to be W[1]-hard by a reduction from p-CLIQUE. However, for restricted $\mathcal{H}$, the problem becomes fixed-parameter tractable. The best results so far are by Chen and Müller [10] who show that when $\mathcal{H}$ has bounded tree-depth, p-EMB($\mathcal{H}$) $\in$ para-L; when $\mathcal{H}$ has bounded path-width, p-EMB($\mathcal{H}$) is the para-L-reduction closure of the distance problem in graphs, parameterized by the distance; and when $\mathcal{H}$ has bounded tree-width, p-EMB($\mathcal{H}$) is the para-L-reduction closure of the embedding problem for trees, parameterized by the tree-size. In contrast to these results, Amano showed for the unparameterized setting, in which we consider the size of H to be a constant, that the problem can be solved in AC^0 with similar techniques [3]. We improve considerably on the first result of Chen and Müller by proving that embeddings of graphs of bounded tree-depth can actually be computed in para-AC^0. We complement their other results, without improving them, by showing that for graphs of bounded tree-width (and, thereby, also for bounded path-width) the embedding problem lies in para-$\mathrm{AC}^{0\uparrow}$.

In order to formulate our results, we first need to review the definition of a tree-decomposition, see [17] for a more detailed introduction. A *tree-decomposition* of a graph $H = (V, E)$ is a tree T together with a mapping ι from the nodes of T to subsets (called *bags*) of V such that (1) for every edge $\{u, v\} \in E$ there is some bag containing u and v, that is, there is some $x \in V$ with $\{u, v\} \subseteq \iota(x)$ and (2) for every vertex $x \in V$ the set of nodes of T whose bags contain x forms a connected subset of T. The *width* of tree-decomposition is the size of its largest bag minus 1, its depth is the maximum of the width and the depth of T. Define $\mathrm{tw}(H)$ as the minimum width any tree-decomposition of H must have; define $\mathrm{td}(H)$ similarly for the tree-depth.

▶ **Theorem 3.5.** *Given two graphs $H = (V_H, E_H)$ and $G = (V_G, E_G)$ together with a tree-decomposition (T, ι) of H. An embedding of H into G can be computed by an* AC*-circuit of depth $O\big(\mathrm{depth}(T)\big)$ and size $f(|V_H|) \cdot O\big(|V_G|^{\mathrm{width}(T)}\big)$, if such an embedding exists.*

Sketch of Proof. Color the vertices of H uniquely and compute a $(|V_G|, |V_H|, |V_H|)$-universal coloring family. Starting from the leaves of the tree-decomposition, merge compatible partial

homomorphisms for the vertices of the bags until we reach the root of the decomposition, and, thus, obtain a homomorphism for H. The number of iterative steps required for this equals the depth of the tree-decomposition. ◀

If H is a parameter, we can compute a width- or depth-bounded tree-decomposition (T, ι) of H in a preprocessing step. This implies the following corollaries:

▶ **Corollary 3.6.** *Let $\mathcal{H}$ be the class of all graphs of tree-width at most d for some constant d. Then p-EMB$(\mathcal{H}) \in$ para-AC$_{f(|H|)} \subseteq$ para-AC$^{0\uparrow}$.*

▶ **Corollary 3.7.** *Let $\mathcal{H}$ be the class of all graphs of tree-depth at most d for some constant d. Then p-EMB$(\mathcal{H}) \in$ para-AC^0.*

We make two remarks at this point: First, one cannot generalize Theorem 3.5 to clique-width since the embedding problem for cliques, which have clique-width 1, is already hard for W[1]. Second, the theorem and the corollary also hold for relational structures $\mathcal{H}$ and $\mathcal{G}$ and if we bound the tree-width of $\mathcal{H}$'s Gaifman graph. Since paths have tree-width 1, the complexity of one of the canonical problems for color-coding – the p_k-PATH problem – can be determined: p_k-PATH $\in$ para-AC$^{0\uparrow}$. This allows us to give a short proof of the following lemma on the complexity of the distance problem for directed graphs where the distance is the parameter (one can also prove this lemma directly quite easily):

▶ **Lemma 3.8.** p_d-DISTANCE $\in$ para-AC$_{f(d)} \subseteq$ para-AC$^{0\uparrow}$.

A known fact from circuit complexity states that a polynomial-sized AC-circuit that decides whether a given graph G contains a path of length at most d between to vertices s and t requires depth $\Omega(\log \log d)$ [5]. This implies p_d-DISTANCE $\notin$ para-AC^0.

▶ **Corollary 3.9.** para-AC$^0 \subsetneq$ para-AC$^{0\uparrow}$.

3.3 First-Order Model Checking

Our last result in this section on tools is an algorithmic meta-theorem: We show that the model checking problem for first-order logic on graphs of bounded degree lies in para-AC$^{0\uparrow}$. We build strongly on a previous result by Flum and Grohe [16], who showed that this model checking problem lies in para-L, but differ in three regards: First, we use color coding in our proof, which simplifies the argument somewhat, second, we identify the parameterized distance problem on bounded degree graphs as the only part of the computation that is presumably not in para-AC^0, and, third, we observe that the degree of the graphs can be made a parameter and need not be considered constant.

▶ **Problem 3.10** ($p_{\phi,\delta}$-MC(FO))**.**
Instance: *A logical structure $\mathcal{A}$ and a first-order formula ϕ.*
Parameter: *The (size of) the formula ϕ and the maximum degree δ of $\mathcal{A}$'s Gaifman graph.*
Question: *$\mathcal{A} \models \phi$?*

▶ **Theorem 3.11.** $p_{\phi,\delta}$-MC(FO) $\in$ para-AC$_{f(\phi+\delta)} \subseteq$ para-AC$^{0\uparrow}$.

Sketch of Proof. By Gaifman's Theorem [20] we can rewrite the given formula as a formula ϕ' in Gaifman normal form. Thus, what essentially remains is to check whether the structure (which we can interpret as a graph) contains k disjoint "balls" of size bounded in the parameter (due to the maximum degree of the underlying Gaifman graph) that satisfy the subformulas in ϕ'. To find these substructures, we make use of color coding and apply Lemma 3.8 to compute the corresponding connecting components. Finally, we only have to model check the resulting parameter-sized substructures. ◀

We conclude with the remark that the depth of the circuits constructed in the above theorem just depends on the degree δ of the graph and on the radius r of the balls, which measure how "local" the formula ϕ is. The smallest r for which ϕ can be rewritten as in the proof is known as the *locality rank* $\mathrm{lr}(\phi)$ and the proof actually shows that $p_{\phi,\delta}$-MC(FO) $\in$ para-$\mathrm{AC}_{O(\delta^{\mathrm{lr}(\phi)})}$.

4 Fast Parallel Fixed-Parameter Algorithms for Natural Problems

The tools we have developed are now applied to a number of natural parameterized problems found in the literature.

Packing Problems. We have already pointed out that the parameterized matching problem can be seen as an embedding problem, where the objective is to embed the graph $H = kK_2$, consisting of k disjoint copies of a single edge, into a graph G. Embedding multiple disjoint copies of the same graph into another graph is also known as "packing". Clearly, instead of edges we can also pack other things as long as taking any number of copies of these "other things" still has bounded tree-depth. For instance, we can try to "pack" k different triangles into G, that is, we can check whether there are k vertex-disjoint triangles in G. Unlike the matching problem, triangle packing is known to be NP-complete.

▶ **Theorem 4.1.** p-TRIANGLE-PACKING $\in$ para-AC^0.

Proof. Just observe that a graph H consisting of any number of disjoint copies of a triangle has tree-depth 3. The claim follows from Corollary 3.7. ◀

Indeed, for any fixed graph H_0 the packing problem p-H_0-PACKING lies in para-AC^0, where the question is whether we can find k disjoint copies of H_0 in G and k is the parameter:

▶ **Theorem 4.2.** p-H_0-PACKING $\in$ para-AC^0 *for every fixed graph* H_0.

Further variants arise when, instead of a single graph H_0, we are given a whole multiset of graphs as inputs and we must find disjoint copies of all of them in G. Again, as long as there is a fixed bound on the size of the graphs, the tree-depth of their disjoint union is bounded and, hence, the packing problem lies in para-AC^0.

The complexity of packing problem changes when the to-be-packed graphs no longer have constant size as in the following problem:

▶ **Problem 4.3** ($p_{k,l}$-CYCLE-PACKING)**.**
Instance: *An undirected graph G and two numbers k and l.*
Parameter: *k and l*
Question: *Are there k vertex-disjoint cycles in G, each having length l?*

The graph $H = kC_l$ consisting of k copies of a cycle of length l no longer has bounded tree-depth; it does have tree-width 2, however. Thus, by Theorem 3.5 we get:

▶ **Theorem 4.4.** $p_{k,l}$-CYCLE-PACKING $\in$ para-$\mathrm{AC}_{f(k+l)} \subseteq$ para-$\mathrm{AC}^{0\uparrow}$.

The same result obviously also holds for $p_{k,l}$-PATH-PACKING and it also holds for p-FOREST-PACKING, where we are given a forest as input and the parameter is the total numbers of vertices in it. We conclude with the remark that these ideas cannot be extended to packing graphs whose tree-width is not bounded: Already embedding cliques, let alone packing them, is W[1]-hard.

Covering Problems. In *covering problems* we must choose vertices in a graph (or sometimes hypergraph) such that all (p-VERTEX-COVER) or some (p-PARTIAL-VERTEX-COVER) of the edges are "covered," that is, they intersect with the set of chosen vertices. The best-known covering problem is undoubtedly p-VERTEX-COVER, whose complexity has been scrutinized extensively in parameterized complexity theory. We now prove p-VERTEX-COVER $\in$ para-AC^0; a fact that nicely reflects on a theoretical basis the "empirical" observation that p-VERTEX-COVER is one of the "easiest" parameterized problems. The problem was one of the first shown to lie in para-P, was then shown to lie in para-L by Cai et al. [8], then in para-TC^0 by Elberfeld and the last two authors [14].

▶ **Theorem 4.5.** *p-VERTEX-COVER $\in$ para-AC^0.*

Partial covering problems ask us not to cover all edges, but only t of them:

▶ **Problem 4.6** ($p_{k,t}$-PARTIAL-VERTEX-COVER)**.**

Instance: *An undirected graph $G = (V, E)$ and two numbers k and t.*
Parameter: *k, t*
Question: *Is there a set $S \subseteq V$ of cardinality $|S|$ at most k such that the cardinality of $\big\{\{u, v\} \in E \mid u \in S \vee v \in S\big\}$ is at least t?*

Another version is p_t-EXACT-PARTIAL-VERTEX-COVER, where the size of S is no longer restricted, but the cardinality of $\big\{\{u, v\} \in E \mid u \in S \vee v \in S\big\}$ must be *exactly* t.

These problems, which are generally considered to be harder than the plain vertex cover problem, lie in the class para-AC$^{0\uparrow}$. Our proofs make an interesting use of Theorem 3.11. Recall that this "meta-theorem" states that all first-order properties of graphs, parameterized by the first-order property and the maximum degree of the graph, can be decided in para-AC$^{0\uparrow}$. Covering properties can be expressed using first-order formulas – but we make *no* requirement concerning the degree of the input graph. The trick is to first reduce the inputs to graphs of bounded degree and then apply the meta theorem. Such a two-step approach is typically in advanced applications of algorithmic meta-theorems.

▶ **Theorem 4.7.** *$p_{k,t}$-PARTIAL-VERTEX-COVER $\in$ para-AC$_{f(k+t)} \subseteq$ para-AC$^{0\uparrow}$.*

▶ **Theorem 4.8.** *p_t-EXACT-PARTIAL-VERTEX-COVER $\in$ para-AC$_{f(t)} \subseteq$ para-AC$^{0\uparrow}$.*

We conclude with the remark that the above results on finding vertex coverings for graphs cannot easily be extended to hypergraphs since for hypergraphs covering problems are typically hard for at least W[1].

Clustering Problems. Clustering algorithms have a wide variety of applications, for example in computational biology where we want to cluster genes and proteins or process transcription data [7]. A basic clustering problem for graphs is the following:

▶ **Problem 4.9** ($p_{k,\ell}$-CLUSTER-EDITING)**.**

Instance: *An undirected graph $G = (V, E)$ and a numbers ℓ and k.*
Parameter: *ℓ, k*
Question: *Can we add and/or delete up to k edges to or from G such that the resulting graph consists of ℓ connected components, each of which is a clique?*

A variant is p_k-MANY-CLUSTER-EDITING, where we just require that the edited graph consists of cliques and do not prescribe the number of clusters beforehand. This variant has been extensively studied, most notably by Gramm et al. [21] and Böcker [6] who showed its

fixed-parameter tractability. For the first version, algorithms based on color coding result in reasonable running times, but where recently be outperformed by other approaches [19]. However, using a color coding approach is useful when we consider parallel algorithms:

▶ **Theorem 4.10.** $p_{k,\ell}$-CLUSTER-EDITING $\in$ para-AC^0.

▶ **Corollary 4.11.** p_k-MANY-CLUSTER-EDITING $\in$ para-AC^0.

We remark that if ℓ is not no longer considered a parameter in cluster editing, the problem complexity increases only moderately:

▶ **Corollary 4.12.** p_k-CLUSTER-EDITING $\in$ para-TC^0.

Theorem 4.10 has another interesting corollary: Let $p_{k,p}$-COMPLETE-p-PARTITE-EDITING be the problem of determining whether in a graph G we can add and / or remove up to k edges such that the resulting graph is complete p-partite, that is, its vertex set can be partitioned into exactly p non-empty sets such that there is an edge between two vertices if, and only if, they belong to two different sets. Since the complement of a complete p-partite graph is exactly a collection of p cliques, we get the following corollary:

▶ **Corollary 4.13.**
1. $p_{k,p}$-COMPLETE-p-PARTITE-EDITING $\in$ para-AC^0.
2. p_k-COMPLETE-p-PARTITE-EDITING $\in$ para-TC^0.

Finally, instead of looking for just one complete p-partite graph, we can look for several at the same time:

▶ **Problem 4.14** ($p_{k,p}$-MULTIPARTITE-CLUSTER-EDITING).

Instance: *An undirected graph $G = (V, E)$, a natural number k, and a sequence of natural numbers $p_1, p_2, \ldots, p_\ell$.*
Parameter: *k, $p = p_1 + \cdots + p_\ell$*
Question: *Can we add or delete k edges of G such that the resulting graph consist of d connected components C_1 to C_ℓ such that each C_i is a complete p_i-partite graph?*

▶ **Theorem 4.15.**
1. $p_{k,p}$-MULTIPARTITE-CLUSTER-EDITING $\in$ para-AC^0
2. $p_{k,\ell}$-MULTIPARTITE-CLUSTER-EDITING $\in$ para-TC^0.

Graph Separation Problems. Graph separation problems are problems where we ask to separate a set of ℓ vertices from the remaining graph by deleting at most k other vertices. They play a key role in many real-world network applications like finding communities or isolating dangerous vertices. While this problem is well-known to be NP-complete in the unparameterized setting and W[1]-hard in the parameterized setting for parameters k, ℓ, and $k + \ell$, the complexity of the problem changes dramatically if we require the separated set of vertices to be connected:

▶ **Problem 4.16** ($p_{k,\ell}$-CUTTING-ℓ-CONNECTED-VERTICES).

Instance: *An undirected graph $G = (V, E)$ and two natural numbers k and ℓ.*
Parameter: *k, ℓ*
Question: *Is there a partitioning of V into three sets X, S, and Y with $|X| = \ell$ and $|S| \leq k$ such that X is connected and for all $\{x, y\} \in E$ with $x \in X$ we have $y \notin Y$?*

Marx [22] showed that this problem is fixed-parameter tractable; Fomin, Golovach, and Korhonen [18] studied a similar version, namely $p_{k,\ell}$-CUTTING-AT-MOST-ℓ-VERTICES, in which the set X is not required to be connected and may be of size *at most* ℓ, i. e., $1 < |X| \leq \ell$, and for which Fomin et al. gave an FPT-algorithm based on color coding. The main idea is to colorize the given graph with two colors such that the vertices of the set X get colored with the first color and the vertices in S get the second color. Thus, we only have to find the solution within the vertices of the first color. This algorithm can be implemented in para-$\mathrm{AC}^{0\uparrow}$ and, moreover, works for $p_{k,\ell}$-CUTTING-ℓ-CONNECTED-VERTICES as well.

▶ **Theorem 4.17.**

1. $p_{k,\ell}$-CUTTING-ℓ-CONNECTED-VERTICES $\in$ para-$\mathrm{AC}_{f(\ell)} \subseteq$ para-$\mathrm{AC}^{0\uparrow}$.
2. $p_{k,\ell}$-CUTTING-AT-MOST-ℓ-VERTICES $\in$ para-$\mathrm{AC}_{f(\ell)} \subseteq$ para-$\mathrm{AC}^{0\uparrow}$.

We conclude with the remark that both problems can also be solved with algorithms similar to the ones presented above if we consider the *terminal versions* of these problems [18], i. e., there is a special terminal vertex t which has to be part of X. For this, we have to modify the above algorithms to consider only blue components that contain t.

5 Conclusion

We have seen that many natural parameterized problems can be solved in constant parallel time or in parallel time depending only on the parameters while doing only "FPT work." We stress that our results are of a theoretical nature and do not directly give practical parallel implementations for the problems presented; but they show that such implementations are possible *in principle* for them. The core technique used in all proofs (at least indirectly) was *color coding,* which can be done in constant time and which is already used in practice.

This paper did not address lower bounds. While for para-AC^0 this is not problematic since this class lies at the bottom of almost any hierarchy of parameterized classes, some problems in para-$\mathrm{AC}^{0\uparrow}$ might well "fall down" to para-AC^0. Here we only know a explicit lower bound for the distance problem, which does not lie in para-AC^0. Establishing lower bounds for other problems in para-$\mathrm{AC}^{0\uparrow}$ is therefore a reasonable research goal.

References

1 F. N. Abu-Khzam, M. A. Langston, P. Shanbhag, and C. T. Symons. Scalable Parallel Algorithms for FPT Problems. *Algorithmica*, 45(3):269–284, 2006.

2 N. Alon, R. Yuster, and U. Zwick. Color-Coding. *Journal of the ACM*, 42(4):844–856, 1995.

3 Kazuyuki Amano. k-Subgraph Isomorphism on AC^0 Circuits. *Computational Complexity*, 19(2):1016–3328, 2010.

4 David A. Mix Barrington and Neil Immerman. On uniformity within NC^1. *Journal of Computer and System Sciences*, 41(3):274–306, 1990.

5 Paul Beame, Russell Impagliazzo, and Toniann Pitassi. Improved depth lower bounds for small distance connectivity. *Computational Complexity*, 7(4):325–345, 1998.

6 S. Böcker. A Golden Ratio Parameterized Algorithm for Cluster Editing. In *Proceedings of the Twenty-Second International Workshop on Combinatorial Algorithms*, volume 7056 of *IWOCA'11*, pages 85–95. Springer Berlin Heidelberg, 2011.

7 S. Böcker and J. Baumbach. Cluster Editing. In *Proceedings of the Ninth Conference on Computability in Europe*, volume 7921 of *CiE'13*, pages 33–44. Springer Berlin Heidelberg, 2013.

8 L. Cai, J. Chen, R. G. Downey, and M. R. Fellows. Advice Classes of Parameterized Tractability. *Annals of Pure and Applied Logic*, 84(1):119–138, 1997.

9 Marco Cesati and Miriam Di Ianni. Parameterized parallel complexity. In *Proceedings of the Fourth International Euro-Par Conference*, volume 1470 of *Euro-Par'89*, pages 892–896. Springer Berlin Heidelberg, 1998.

10 H. Chen and M. Müller. The Fine Classification of Conjunctive Queries and Parameterized Logarithmic Space. *ACM Transactions on Computation Theory*, 7(2):7:1–7:27, 2014.

11 Y. Chen, J. Flum, and M. Grohe. Bounded Nondeterminism and Alternation in Parameterized Complexity Theory. In *Proceedings of the Eighteenth IEEE Conference on Computational Complexity*, CCC'03, pages 13–29. IEEE Computer Society, Los Alamitos, California, 2003.

12 R.G. Downey and M.R. Fellows. *Parameterized Complexity*. Springer, 1999.

13 Rodney G. Downey, Michael R. Fellows, and Kenneth W. Regan. Parameterized Circuit Complexity and the W Hierarchy. *Theoretical Computer Science*, 191(1–2):97–115, 1998.

14 M. Elberfeld, C. Stockhusen, and T. Tantau. On the Space Complexity of Parameterized Problems: Classes and Completness. *Algorithmica*, 71(3):661–701, 2014.

15 M. Fellows, P. Heggernes, F. Rosamond, C. Sloper, and J. A. Telle. Finding k Disjoint Triangles in an Arbitrary Graph. In *Proceedings of the Thirtieth International Workshop on Graph-Theoretic Concepts in Computer Science*, volume 3353 of *WG'04*, pages 235–244. Springer Berlin Heidelberg, 2004.

16 J. Flum and M. Grohe. Describing Parameterized Complexity Classes. In *Proceedings of the Nineteenth Annual Symposium on Theoretical Aspects of Computer Science*, volume 2285 of *STACS'02*, pages 359–371. Springer Berlin Heidelberg, 2002.

17 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.

18 F. V. Fomin, P. A. Golovach, and J. H. Korhonen. On the Parameterized Complexity of Cutting a Few Vertices from a Graph. In *Proceedings of the Thirty-Eight International Symposium of Mathematical Foundations of Computer Science*, volume 8087 of *MFCS '13*, pages 421–432. Springer, Heidelberg, Germany, 2013.

19 F. V. Fomin, S. Kratsch, M. Pilipczuk, M. Pilipczuk, and Y. Villanger. Tight Bounds for Parameterized Complexity of Cluster Editing. In *Proceedings of the Thirtieth International Symposium on Theoretical Aspects of Computer Science*, volume 20 of *STACS'13*, pages 30–43. International Symposium on Theoretical Aspects of Computer Science, 2013.

20 H. Gaifman. On Local and Non-Local Properties. In *Proceedings of the Herbrand Symposium, Logic Colloquium 1981*, pages 105–135. North Holland, 1982.

21 J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Graph-Modeled Data Clustering: Fixed-Parameter Algorithms for Clique Generation. In *Proceedings of the Fifth Italian Conference of Algorithms and Complexity*, volume 2653 of *CIAC'03*, pages 108–119. Springer Berlin Heidelberg, 2003.

22 D. Marx. Parameterized Graph Separation Problems. *Theoretical Computer Science*, 351(3):394–406, 2006.

23 I. Newman, P. Ragde, and A. Wigderson. Perfect Hashing, Graph Entropy, and Circuit Complexity. In *Proceedings of the Fifth Annual Structure in Complexity Theory Conference*, pages 91–99. IEEE Computer Society, Los Alamitos, California, 1990.

24 H. Vollmer. *Introduction to Circuit Complexity*. Springer, 1999.

Fixed-parameter Tractable Distances to Sparse Graph Classes

Jannis Bulian and Anuj Dawar

University of Cambridge Computer Laboratory, UK
firstname.lastname@cl.cam.ac.uk

Abstract

We show that for various classes $\mathcal{C}$ of sparse graphs, and several measures of distance to such classes (such as edit distance and elimination distance), the problem of determining the distance of a given graph G to $\mathcal{C}$ is fixed-parameter tractable. The results are based on two general techniques. The first of these, building on recent work of Grohe et al. establishes that any class of graphs that is slicewise nowhere dense and slicewise first-order definable is FPT. The second shows that determining the elimination distance of a graph G to a minor-closed class $\mathcal{C}$ is FPT.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases parameterized complexity, fixed-parameter tractable, distance, graph theory, sparse graphs, graph minor, nowhere dense

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.236

1 Introduction

The study of parameterized algorithmics for graph problems has thrown up a large variety of structural parameters of graphs. Among these are parameters that measure the *distance* of a graph G to a class $\mathcal{C}$ in some way. The simplest such measures are those that count the number of vertices or edges that one must delete (or add) to G to obtain a graph in $\mathcal{C}$. A common motivation for studying such parameters is that if a problem one wishes to solve is tractable on the class $\mathcal{C}$, then the distance to $\mathcal{C}$ provides an interesting parameterization of that problem (called *distance to triviality* by Guo et al. [12]). Other examples of this include the study of modulators to graphs of bounded tree-width in the context of kernelization (see [8, 9]) or the parameterizations of colouring problems (see [15]). On the other hand, determining the distance of an input graph G to a class $\mathcal{C}$ is, in general, a computationally challenging problem in its own right. Such problems have also been extensively studied with a view to establishing their complexity when parameterized by the distance. A canonical example is the problem of determining the size of a minimum vertex cover in a graph G, which is just the vertex-deletion distance of G to the class of edge-less graphs. More generally, Cai [3] studies the parameterized complexity of distance measures defined in terms of addition and deletion of vertices and edges to hereditary classes $\mathcal{C}$. Counting deletions of vertices and edges gives a rather simple notion of distance, and many more involved notions have also been studied. Classic examples include the crossing number of a graph which provides one notion of distance to the class of planar graphs or the treewidth of a graph which can be seen as a measure of distance to the class of trees. Another recently introduced measure is *elimination distance*, defined in [2] where it was shown that graph isomorphism is FPT when parameterized by elimination distance to a class of graphs of bounded degree.

In this paper we consider the fixed-parameter tractability of a variety of different notions of distance to various different classes $\mathcal{C}$ of sparse graphs. We establish two quite general

techniques for establishing that such a distance measure is FPT. The first builds on the recent result of Grohe et al. [11] which shows that the problem of evaluating first-order formulas on any *nowhere dense* class of graphs is FPT with the formula as parameter. We extract from their proof of this result a general statement about the fixed-parameter tractability of definable sparse classes. To be precise, we show that parameterized problems that are both *slicewise nowhere dense* and *slicewise first-order definable* (these terms are defined precisely below) are FPT. As an application of this, it follows that if $\mathcal{C}$ is a nowhere dense class of graphs that is definable by a first-order formula, then the parameterized problem of determining the distance of a graph G to $\mathcal{C}$ is FPT, for various notions of distance that can be themselves so defined. In particular, we get that various forms of edit distance to classes of bounded-degree graphs are FPT (a result established by Golovach [10] by more direct methods). Another interesting application is obtained by considering elimination distance of a graph G to the class $\mathcal{C}$ of empty graphs. This is nothing other than the *tree-depth* of G. While elimination distance to a class $\mathcal{C}$ is in general not first-order definable, it is in the particular case where $\mathcal{C}$ is the class of empty graphs. Thus, we obtain as an application of our method the result that tree-depth is FPT, a result previously known from other algorithmic meta theorems (see [16, Theorem 17.2]). The method of establishing that a parameterized problem is FPT by establishing that it is slicewise nowhere dense and slicewise first-order definable appears to be a powerful method of some generality which will find application beyond these examples.

Our second general method specifically concerns elimination distance to a minor-closed class $\mathcal{C}$. We show that this measure is fixed-parameter tractable for any such $\mathcal{C}$, answering an open question posed in [2]. Note that while a proper minor-closed class is always nowhere dense, it is not generally first-order definable (for instance, neither the class of acyclic graphs nor the class of planar graphs is), and elimination distance to such a class is also not known to be first-order definable. Thus, our results on the tractability of slicewise first-order definable classes do not apply here. Instead, we build on work of Adler et al. [1] to show that from a finite list of the forbidden minors characterising $\mathcal{C}$, we can compute the set of forbidden minors characterising the graphs at elimination distance k to $\mathcal{C}$. Adler et al. show how to do this for apex graphs, from which one immediately obtains the result for graphs that are k deletions away from $\mathcal{C}$. To extend this to elimination distance k, we show how we can construct the forbidden minors for the closure of a minor-closed class under disjoint unions.

In Section 2 we present the definitions necessary for the rest of the paper. Section 3 establishes our result for slicewise first-order definable and slicewise nowhere dense problems and gives some applications of the general method. Section 4 establishes that the problem of determining elimination distance to any minor-closed class is FPT. Some open questions are discussed in Section 5. Due to limitations of space, some material is deferred to the full version of this paper, which may be found at `arxiv:1502.05910`.

2 Preliminaries

First-order logic

We assume some familiarity with first-order logic for Section 3. A *(relational) signature* σ is a finite set of relation symbols, each with an associated arity. A *σ-structure* A consists of a set $V(A)$ and for each k-ary relation symbol $R \in \sigma$ a relation $R(A) \subseteq V(A)^k$. Our structures are mostly (coloured) graphs, so $\sigma = \{E\}$ or $\sigma = \{E, C_1, C_2, \ldots, C_r\}$ where E is binary and the C_i are unary relation symbols. A graph G is then a σ-structure with vertex set $V(G)$, edge relation $E(G)$, and colours $C_i(G)$.

A first-order formula φ is recursively defined by the following rules:

$$\varphi := R(x_1, \ldots, x_r) \mid x = y \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x.\varphi.$$

We also use the following abbreviations:

$$\varphi \wedge \psi := \neg(\neg\varphi \vee \neg\psi), \qquad \forall x.\varphi := \neg\exists.\neg\varphi.$$

The *quantifier rank* of a formula φ is the nesting depth of quantifiers in φ. For a more detailed presentation we refer to Hodges [13].

Parameterized Complexity

Parameterized complexity theory is a two-dimensional approach to the study of the complexity of computational problems. We find it convenient to define problems as classes of structures rather than strings. A *problem* $Q \subseteq str(\sigma)$ is an (isomorphism-closed) class of σ-structures given some signature σ. A *parameterization* is a computable function $\kappa : str(\sigma) \to \mathbb{N}$. We say that Q is *fixed-parameter tractable* with respect to κ if we can decide whether an input $A \in str(\sigma)$ is in Q in time $O(f(\kappa(A)) \cdot |A|^c)$, where c is a constant and f is some computable function. For a thorough discussion of the subject we refer to the books by Downey and Fellows [5], Flum and Grohe [7] and Niedermeier [17].

A parameterized problem (Q, κ) is *slicewise first-order definable* if there is a computable function $f : \mathbb{N} \to \mathrm{FO}[\sigma]$ such that a σ-structure A with $\kappa(A) \leq i$ is in Q if, and only if, $A \models f(i)$. Slicewise definability of problems in a logic was introduced by Flum and Grohe [6].

Graph theory

A *graph* G is a set of vertices $V(G)$ and a set of edges $E(G) \subseteq V(G) \times V(G)$. We assume that graphs are loop-free and undirected, i.e. that E is irreflexive and symmetric. We mostly follow the notation in Diestel [4]. For a set $S \subseteq V(G)$ of vertices, we write $G \setminus S$ to denote the subgraph of G induced by $V(G) \setminus S$.

Let $r \in \mathbb{N}$. An r-independent set in a graph G is a set of vertices of G such that their pairwise distance is at least r.

A graph H is a *minor* of a graph G, written $H \preceq G$, if there is a map, called the *minor map*, that takes each vertex $v \in V(H)$ to a tree T_v that is a subgraph of G such that for any $u \neq v$ the trees are disjoint, i.e. $T_v \cap T_u = \emptyset$, and such that for every edge $uv \in E(H)$ there are vertices $u' \in T_u, v' \in T_v$ with $u'v' \in E(G)$. A class of graphs $\mathcal{C}$ is *minor-closed* if $H \preceq G \in \mathcal{C}$ implies $H \in \mathcal{C}$.

The *set of minimal excluded minors* $M(\mathcal{C})$ is the set of graphs in the complement of $\mathcal{C}$ such that for each $G \in M(\mathcal{C})$ all proper minors of G are in $\mathcal{C}$. By the Robertson-Seymour Theorem [18] the set $M(\mathcal{C})$ is finite for every minor-closed class $\mathcal{C}$. It is a consequence of this theorem that membership in a minor-closed class can be tested in $O(n^3)$ time. For a set M of graphs, we write $\mathrm{Forb}(M)$ for the class of graphs which forbid M as minors, i.e. $\mathrm{Forb}(M) = \{G \mid H \not\preceq G \text{ for all } H \in M\}$.

Let $r \in \mathbb{N}$. A minor H of G is a *depth-r minor of* G, written $H \preceq_r G$, if there is a minor map that takes vertices in H to trees that have radius at most r. A class of graphs $\mathcal{C}$ is *nowhere dense* if for every $r \in \mathbb{N}$ there is a graph H_r such that for no $G \in \mathcal{C}$ we have $H_r \preceq_r G$. A nowhere-dense class of graphs $\mathcal{C}$ is called *effectively nowhere dense* if there is a computable function f from integers to graphs such that for no $G \in \mathcal{C}$ and no r we have $f(r) \preceq_r G$. We are only interested in effectively nowhere-dense classes so we simply use the term *nowhere dense* to mean effectively nowhere dense.

We say that a parameterized graph problem (Q, κ) is *slicewise nowhere dense* if there is a computable function h from pairs of integers to graphs such that for all $i \in \mathbb{N}$, we have for no $G \in \{H \in Q \mid \kappa(H) \leq i\}$ and r that $h(i, r) \preceq_r G$. We will call h the *parameter function* of Q.

For a class of graphs $\mathcal{C}$ we denote the closure of $\mathcal{C}$ under taking disjoint unions by $\overline{\mathcal{C}}$. We say that a graph G is an *apex graph* over a class $\mathcal{C}$ of graphs if there is a vertex $v \in V(G)$ such that the graph $G \setminus \{v\} \in \mathcal{C}$. The class of all apex graphs over $\mathcal{C}$ is denoted $\mathcal{C}^{\text{apex}}$.

A graph G has *deletion distance k to a class $\mathcal{C}$* if there are k vertices $v_1, \ldots, v_k \in V(G)$ such that $G \setminus \{v_1, \ldots, v_k\} \in \mathcal{C}$.

The *elimination distance* of a graph G to a class $\mathcal{C}$ is defined as follows:

$$ed_{\mathcal{C}}(G) := \begin{cases} 0, & \text{if } G \in \mathcal{C}; \\ 1 + \min\{ed_{\mathcal{C}}(G \setminus v) \mid v \in V(G)\}, & \text{if } G \notin \mathcal{C} \text{ and } G \text{ is connected;} \\ \max\{ed_{\mathcal{C}}(H) \mid H \text{ a connected component of } G\}, & \text{otherwise.} \end{cases}$$

3 A general method for editing distances

In this section we establish a general technique for showing that certain definable parameterized problems on graphs are FPT. As an application, we show that certain natural distance measures to sparse graph classes are FPT. To be precise, we show that if a parameterized problem is both slicewise first-order definable and slicewise nowhere dense, then it is FPT. In particular, this implies that if we have a class $\mathcal{C}$ that is first-order definable and nowhere dense and the distance measure we are interested in is also first-order definable (that is to say, for each k there is a formula that defines the graphs of distance k from $\mathcal{C}$), then the problem of determining the distance is FPT. More generally, if we have a parameterized problem (Q, κ) that is slicewise nowhere dense and slicewise first-order definable, and a measure of distance to it is definable in the sense that for any values of k and d, there is a first-order formula defining the graphs of distance d to the class $\{G \mid G \in Q \text{ and } \kappa(G) \leq k\}$, then the problem of deciding whether a graph has distance at most d to this class is FPT parameterized by $d + k$. In particular, this yields the result of Golovach [10] as a consequence.

The method is an adaption of the main algorithm in Grohe *et al.* [11]. Since the proof is essentially a modification of their central construction, rather than give a full account, we state the main results they prove and explain briefly how the proofs can be adapted for our purposes. For a full proof, this section is best read in conjunction with the paper [11]. Section 3.1 gives an overview of the key elements of the construction from [11] and the elements from it which we need to extract for our result. Section 3.2 then gives our main result and Section 3.3 derives some consequences for distance measures.

3.1 Evaluating Formulas on Nowhere Dense Classes

The key result of [11] is:

▶ **Theorem 1** (Grohe *et al.* [11, Theorem 1.1]). *For every nowhere dense class $\mathcal{C}$ and every $\epsilon > 0$, every property of graphs definable in first-order logic can be decided in time $O(n^{1+\epsilon})$ on $\mathcal{C}$.*

In the full version of the paper we give a sketch of the algorithm from Theorem 1 with an emphasis on the changes needed for our purposes. Here we state the main results that we extracted and that are needed for the next section.

A key data structure used in the algorithm is a neighbourhood cover. An important result from [11] is that graphs from a nowhere dense class allow for small covers and that such a cover can be efficiently computed.

▶ **Theorem 2** (Grohe *et al.* [11, Theorem 6.2])**.** *Let $\mathcal{C}$ be a nowhere dense class of graphs. There is a function f such that for all $r \in \mathbb{N}$ and $\epsilon > 0$ and all graphs $G \in \mathcal{C}$ with $n \geq f(r, \epsilon)$ vertices, there exists an r-neighbourhood cover of radius at most $2r$ and maximum degree at most n^ϵ and this cover can be computed in time $f(r, \epsilon) \cdot n^{1+\epsilon}$. Furthermore, if $\mathcal{C}$ is effectively nowhere dense, then f is computable.*

While the algorithm of [11] assumes that the input graph G comes from the class $\mathcal{C}$, we can say something more. For a fixed nowhere dense class $\mathcal{C}$, where we know the parameter function h, we can, given G, r and ϵ, compute a bound on the running time of the algorithm from Theorem 2 . By running the algorithm to this bound, we have the following as a direct consequence.

▶ **Lemma 3.** *Let $\mathcal{C}$ be a nowhere dense class of graphs. There is a function f such that for all $r \in \mathbb{N}$ and $\epsilon > 0$ and all graphs $G \in \mathcal{C}$ with $n \geq f(r, \epsilon)$ vertices, there exists an r-neighbourhood cover of radius at most $2r$ and maximum degree at most n^ϵ. There is an algorithm that given an arbitrary graph G runs in time $f(r, \epsilon) \cdot n^{1+\epsilon}$ and that computes this cover or determines that $G \notin \mathcal{C}$. Furthermore, if $\mathcal{C}$ is effectively nowhere dense, then f is computable.*

At the core of the proof of Theorem 1 is the Rank-Preserving Locality Theorem. We state a simplified version here. More details can be found in the full version of the paper.

▶ **Theorem 4** (Rank-Preserving Locality Theorem, Grohe *et al.* [11, Theorem 7.5])**.** *For every $q \in \mathbb{N}$ there is an r such that for every* FO*-formula $\varphi(x)$ of quantifier rank q there is a formula with an extended signature $\hat{\varphi}(x)$ and a graph G' (both depending on q and r) such that for every $v \in V(G)$,*

$$G \models \varphi(v) \iff G' \models \hat{\varphi}(v).$$

Furthermore, $\hat{\varphi}$ is computable from φ, and r is computable from q.

We observe that the structure G' mentioned in Theorem 4 can be efficiently computed:

▶ **Lemma 5.** *Let $\mathcal{C}$ be a nowhere dense class of graphs. There is an algorithm that runs in time $O(q)$ which, given a graph G, returns G' or determines that $G \notin \mathcal{C}$.*

Theorem 4 reduces the problem of evaluating a formula of first-order logic to deciding a series of distance-r-independent set problems. So, the final ingredient is to show that this is tractable. Formally, the problem is defined as follows:

Distance Independent Set
Input: A graph G and $k, r \in \mathbb{N}$.
Parameter: $k + r$
Problem: Does G contain an r-independent set of size k?

The problem is shown to be FPT on nowhere dense classes of graphs [11, Theorem 5.1], and the theorem can be restated as follows:

▶ **Lemma 6.** *Let $\mathcal{C}$ be a nowhere dense class of graphs. Then there is an algorithm and a function f such that for every $\epsilon > 0$ the algorithm runs in time $f(\epsilon, r, k)$ and either solves the* Distance Independent Set *problem or determines $G \notin \mathcal{C}$. If $\mathcal{C}$ is effectively nowhere dense, then f is computable.*

This is all we need to evaluate $\hat{\varphi}$ on G', which is equivalent to evaluating φ on G by Theorem 4.

3.2 Deciding Definable nowhere dense Problems

The main result of [11] establishes that checking whether $G \models \varphi$ is FPT when parameterized by φ provided that G comes from a known nowhere dense class $\mathcal{C}$. Thus, the formula is arbitrary, but the graphs come from a restricted class. Section 3.1 gives an account of this proof from which we can extract the observation that the algorithm can be modified to work for an arbitrary input graph G with the requirement that the algorithm *may* simply reject the input if G is not in $\mathcal{C}$. This suggests a tractable way of deciding $G \models \varphi$ provided that φ defines a nowhere dense class. Now the graph is arbitrary, but the formula comes from a restricted class. We formalise the result in the following theorem:

▶ **Theorem 7.** *Let (Q, κ) be a problem that is slicewise first-order definable and slicewise nowhere dense. Then (Q, κ) is fixed-parameter tractable.*

Proof. In the following, for ease of exposition, we assume that an instance of the problem consists of a graph G and $\kappa(G) = i$ for some positive integer i.

Step 1: Compute φ and the parameters function: Since (Q, κ) is slicewise first-order definable, we can compute from i a first-order formula φ which defines the class of graphs $C_i = \{H \mid H \in Q \text{ and } \kappa(H) \leq i\}$. Moreover, since (Q, κ) is slicewise nowhere dense, we can compute from i an algorithm that computes the parameter function h for C_i.

Step 2: Obtain $\hat{\varphi}$ from φ: By the Rank-Preserving Locality Theorem (Theorem 4), we can compute from φ the formula $\hat{\varphi}$ and a radius r.

Step 3: Find a small cover $\mathcal{X}$ for G: By Lemma 3, we can either find a cover $\mathcal{X}$ for G, or reject if the algorithm determine that $G \notin C_i$.

Step 4: Simulate Splitter game to compute G': By Lemma 5 we obtain G' or reject if the algorithm determines that $G \notin C_i$.

Step 5: Evaluate $\hat{\varphi}$ on G': Finally to evaluate $\hat{\varphi}$ on G', we need to solve the distance independent set problem. We can do this by Lemma 6. Since evaluating $\hat{\varphi}$ on G' is equivalent to evaluating φ on G this allows us to decide whether $G \in Q$. ◀

3.3 Applications

In this Section we discuss some applications of Theorem 7 that demonstrate its power. We begin by considering simple edit distances.

Edit Distances

A graph G has *deletion distance* k to a class $\mathcal{C}$ if there exists a set S of k vertices in G so that $G \setminus S \in \mathcal{C}$. Suppose (Q, κ) is a parameterized graph problem. We define the problem of deletion distance to Q as follows:

Deletion Distance to Q
Input: A graph G and $k, d \in \mathbb{N}$.
Parameter: $k + d$
Problem: Does G contain a set S of k vertices so that $\kappa(G \setminus S) \leq d$ and $G \setminus S \in Q$?

In many of the examples below, we define formulas of first-order logic by *relativisation*. For convenience, we define the notion here.

▶ **Definition 8.** Let φ and $\psi(x)$ be first-order formulas, where ψ has a distinguished free variable x . The relativisation of φ by ψ, denoted $\varphi^{[x.\psi]}$ is the formula obtained from φ by replacing all subformulas of the form $\exists v\varphi'$ in φ by $\exists v(\psi[v/x] \wedge \varphi')$, and all subformulas of the form $\forall v\varphi'$ in φ by $\forall v(\psi[v/x] \to \varphi')$. Here $\psi[v/x]$ denotes the result of replacing the free occurrences of x in ψ with v in a suitable way avoiding capture.

The key idea here is that $\varphi^{[x.\psi]}$ is true in G iff φ is true in the subgraph of G induced by the vertices that satisfy $\psi(x)$. Note that the variable x that is free is ψ is bound in $\varphi^{[x.\psi]}$. Other variables that appear free in ψ remain free in $\varphi^{[x.\psi]}$.

▶ **Proposition 9.** *If (Q, κ) is slicewise nowhere dense and slicewise first-order definable then* Deletion Distance to Q *is* FPT*.*

Proof. It suffices to show that Deletion Distance to Q is also slicewise nowhere dense and slicewise first-order definable. For the latter, note that if φ_i is the first-order formula that defines the class of graphs $\mathcal{C}_i = \{G \mid \kappa(G) \leq i \text{ and } G \in Q\}$, then the class of graphs at deletion distance k to $\mathcal{C}_i$ is given by:

$$\exists w_1, \ldots, w_k \varphi_i^{[x.\theta_k]}$$

where $\theta_k(x)$ is the formula $\bigwedge_{1 \leq i \leq k} x \neq w_i$.

Since $\mathcal{C}_i$ is slicewise nowhere dense, there is a computable function f such that for all i and r the graph $H_i = f(i)$ is not an r-minor of any of the graphs in the class $\mathcal{C}_i$. Observe that if a graph $G \in \mathcal{C}_i$ excludes H_i with $|H_i| = m_i$ vertices as an r-minor, then it also excludes K_{m_i} as an r-minor.

To see that Deletion Distance to Q is also slicewise nowhere dense, note that a graph with deletion distance k to a graph in $G \in \mathcal{C}_i$ cannot contain K_{m_i+k} as an r-minor. We can thus define $g(r, k) = K_{m_i+k}$ as the parameter function of the class of graphs with deletion distance k to $\mathcal{C}_i$. ◀

Instead of deleting vertices, we can also consider editing the graph by adding or deleting edges. It is easily seen that we can modify a first-order formula φ to define the class of graphs G that can be made to satisfy G by k edge additions or deletions. Thus, an analogue of Proposition 9 is obtained for any combination of vertex and edge deletions and additions. Golovach [10] proved that that editing a graph to degree d using at most k edge additions/deletions is FPT parameterized by $k + d$. Since the class of graphs of degree d is first-order definable and nowhere dense for any d, the result also now follows from Theorem 7.

Tree-depth

Tree-depth is a graph parameter that lies between the widely studied parameters vertex cover number and tree width. It has interesting connections to nowhere dense graph classes. It is usually defined as follows:

▶ **Definition 10.** The *tree-depth* of a graph G, written $td(G)$, is

$$td(G) := \begin{cases} 0, & \text{if } V(G) = \emptyset; \\ 1 + \min\{td(G \setminus v) \mid v \in V(G)\}, & \text{if } G \text{ is connected;} \\ \max\{td(H) \mid H \text{ a component of } G\}, & \text{otherwise.} \end{cases}$$

Note that a graph has tree-depth k if, and only if, it has elimination distance k to the class of empty graphs. So one can think of elimination distance as a natural generalisation of tree-depth.

It is known that the problem of determining the tree-depth of graph is FPT, with tree-depth as the parameter (see [16, Theorem 7.2]). We now give an alternative proof of this, using Theorem 7. It is clear that for any k, the class of graphs of tree-depth at most k is nowhere dense. We show below that it is also first-order definable.

▶ **Proposition 11.** *For each $k \in \mathbb{N}$ there is a first-order formula φ_k such that a graph G has tree-depth k if and only if $G \models \varphi_k$.*

Proof. We use the fact that in a graph of tree-depth less than k, there are no paths of length greater than 2^k. This allows us, in the inductive definition of tree-depth above, to replace the condition of connectedness (which is not first-order definable) with a first-order definable condition on vertices at distance at most 2^k.

Let $\mathrm{dist}_d(u, v)$ denote the first-order formula with free variables u and v that is satisfied by a pair of vertices in a graph G if, and only if, they have distance at most d in G. Note that the formula $\mathrm{dist}_d^{[x.x\neq w]}(u, v)$ is then a formula with three free variables u, v, w which defines those u, v which have a path of length d in the graph obtained by deleting the vertex w.

We can now define the formula φ_k by induction. Only the empty graph has tree-depth 0, so $\varphi_0 := \neg\exists v(v = v)$.

Suppose that φ_k defines the graphs of tree-depth at most k, let

$$\theta_k := (\forall u, v \mathrm{dist}_{2^{k+1}}(u, v)) \wedge \exists w(\varphi_k^{[x.x\neq w]}).$$

The formula θ_k defines the connected graphs of tree depth at most $k + 1$. Indeed, the first conjunct ensures that the graph is connected as no pair of vertices has distance greater than 2^{k+1} and that we can find a vertex w whose removal yields a graph of tree-depth at most k.

We can now define the formula φ_{k+1} as follows.

$$\varphi_{k+1} := (\forall u, v \mathrm{dist}_{2^{k+1}+1}(u, v) \to \mathrm{dist}_{2^{k+1}}(u, v)) \wedge \forall w \theta_k^{[x.\mathrm{dist}_{2^{k+1}}(w,x)]}.$$

The formula asserts that there are no pairs of vertices whose distance is strictly greater than 2^{k+1} and that for every vertex w, the formula θ_k holds in its connected component, namely those vertices which are at distance at most 2^{k+1} from w. ◀

While the proof of Proposition 9 shows that deletion distance to any slicewise first-order definable class is also slicewise first-order definable, Proposition 11 shows that elimination distance to the particular class of empty graphs is slicewise first-order definable. It does not establish this more generally for elimination distance to any slicewise nowhere dense class.

4 Elimination distance to classes characterised by excluded minors

In this section we show that determining the elimination distance of a graph to a minor-closed class $\mathcal{C}$ is FPT when parameterized by the elimination distance. More generally, we formulate the following parameterized problem where the forbidden minors of $\mathcal{C}$ are also part of the parameter.

Elimination Distance to Excluded Minors
Input: A graph G, a natural number $k \in \mathbb{N}$ and a set of graphs M
Parameter: $k + \sum_{H \in M} |H|$
Problem: Does G have elimination distance k to the class Forb(M)?

It is not difficult to show that the class of graphs which have elimination distance k to a minor-closed class $\mathcal{C}$ is also minor-closed. Indeed, this can be seen directly from an alternative characterisation of elimination distance that we establish below. The characterisation is in terms of the iterated closure of $\mathcal{C}$ under the operation of disjoint unions and taking the class of apex graphs. We introduce a piece of notation for this in the next definition. Recall that we write $\mathcal{C}^{\text{apex}}$ for the class of all the apex graphs over $\mathcal{C}$, and that we write $\overline{\mathcal{C}}$ for the closure of $\mathcal{C}$ under disjoint unions.

▶ **Definition 12.** For a class of graphs $\mathcal{C}$, let $\mathcal{C}_0 := \mathcal{C}$, and $\mathcal{C}_{i+1} := \overline{\mathcal{C}_i^{\text{apex}}}$.

We show next that the class $\mathcal{C}_k$ is exactly the class of graphs at elimination distance k from $\mathcal{C}$.

▶ **Proposition 13.** *Let $\mathcal{C}$ be a class of graphs and $k \geq 0$. Then $\mathcal{C}_k$ is the class of all graphs with elimination distance at most k to $\mathcal{C}$.*

Proof. We prove this by induction. Only the graphs in $\mathcal{C}$ have elimination distance 0 to $\mathcal{C}$, so the statement holds for $k = 0$.

Suppose the statement holds for k. If $G \in \mathcal{C}_{k+1}$, then G is a disjoint union of graphs $G_1, \ldots, G_s$ from ${\mathcal{C}_k}^{\text{apex}}$, so we can remove at most one vertex from each of the G_i and obtain a graph in $\mathcal{C}_k$. Thus the elimination distance of G to $\mathcal{C}_k$ is 1, and by induction the elimination distance to $\mathcal{C}$ is $k + 1$. Conversely, if G has elimination distance $k + 1$ to $\mathcal{C}$, then we can remove a vertex from each component of G to obtain a graph G' with elimination distance k to $\mathcal{C}$. Using the induction hypothesis each component of G' is in $\mathcal{C}_k$, and thus $G \in \mathcal{C}_{k+1}$. ◀

It is easy to see that if $\mathcal{C}$ is a minor-closed class of graphs then so is $\mathcal{C}_k$ for any k. Indeed, it is well-known that C^{apex} is minor-closed for any minor-closed $\mathcal{C}$, so we just need to note that $\overline{\mathcal{C}}$ is also minor-closed. But it is clear that if H is a minor of a graph G that is the disjoint union of graphs $G_1, \ldots, G_s$, then H itself is the disjoint union of minors of $G_1, \ldots, G_s$. Thus, the class of graphs of elimination distance at most k to a minor-closed class $\mathcal{C}$ is itself minor-closed. We next show that we can construct the set of its minimal excluded minors from the corresponding set for $\mathcal{C}$.

To obtain $M(\mathcal{C}_k)$, we need to iteratively compute $M(\mathcal{C}^{\text{apex}})$ and $M(\overline{\mathcal{C}})$ from $M(\mathcal{C})$. Adler et al. [1] show that from the set of minimal excluded minors $M(\mathcal{C})$ of a class $\mathcal{C}$, we can compute $M(\mathcal{C}^{\text{apex}})$:

▶ **Theorem 14** ([1], Theorem 5.1)**.** *There is a computable function that takes the set of graphs $M(\mathcal{C})$ characterising a minor-closed class $\mathcal{C}$ to the set $M(\mathcal{C}^{apex})$.*

We next aim to show that from $M(\mathcal{C})$ we can also compute $M(\overline{\mathcal{C}})$. Together with Theorem 14 this implies that from $M(\mathcal{C})$ we can compute $M(\mathcal{C}_k)$.

We begin by characterising minor-closed classes that are closed under disjoint unions in terms of the connectedness of their excluded minors.

▶ **Lemma 15.** *Let $\mathcal{C}$ be a class of graphs closed under taking minors. Then $\mathcal{C}$ is closed under taking disjoint unions iff each graph in $M(\mathcal{C})$ is connected.*

Proof. Let $\mathcal{C}$ be a minor-closed class of graphs, and let $M(\mathcal{C})$ be its set of minimal excluded minors.

Suppose each of the graphs in $M(\mathcal{C})$ is connected. Let $H \in M(\mathcal{C})$ and let $G = G_1 \oplus \cdots \oplus G_r$ be the disjoint union of graphs $G_1, \ldots, G_r \in \mathcal{C}$. Because H is connected, we have that $H \preceq G$ if, and only if, $H \preceq G_i$ for some i. So, since for each i, $G_i \in \mathcal{C}$, we have $H \not\preceq G$ and thus $G \in \mathcal{C}$. This shows that $\mathcal{C}$ is closed under taking disjoint unions.

Conversely assume $H \in M(\mathcal{C})$ is not connected and let $A_1, \ldots, A_t$ be its connected components. Then $A_1, \ldots, A_t \in \mathcal{C}$, since each A_i is a proper minor of H, and H is minor-minimal in the complement of $\mathcal{C}$. However, $A_1 \oplus \cdots \oplus A_t = H \notin \mathcal{C}$. ◀

▶ **Definition 16.** For a graph G with connected components $G_1, \ldots, G_r$, let $\mathcal{H}$ denote the set of *connected* graphs H with $V(H) = V(G)$ and such that the subgraph of H induced by $V(G_i)$ is exactly G_i. We define the *connection closure* of G to be the set of all minimal (under the subgraph relation) graphs in $\mathcal{H}$. The connection closure of a set of graphs is the union of the connection closures of the graphs in the set.

Note that if G has e edges and m components, then any graph in the connection closure of G has exactly $e + m - 1$ edges. This is because it has G as a subgraph and in addition $m - 1$ edges corresponding to a tree on m vertices connecting the m components.

▶ **Lemma 17.** *Let $\mathcal{C}$ be a minor-closed class of graphs. Then $M(\overline{\mathcal{C}})$ is the set of minor-minimal graphs in the connection closure of $M(\mathcal{C})$.*

Proof. Let $\mathcal{C}$ be a minor-closed class of graphs, with $M(\mathcal{C})$ its set of minimal excluded minors, and let $\hat{M}$ be the connection closure of $M(\mathcal{C})$.

Let G be a graph such that $\hat{H} \not\preceq G$ for all $\hat{H} \in \hat{M}$. Suppose for contradiction that G is not a disjoint union of graphs from $\mathcal{C}$. Then there is a component G' of G that is not in $\mathcal{C}$ and therefore there is a graph $H \in M(\mathcal{C})$ such that $H \preceq G'$. We show that one of the graphs in the connection closure of H is a minor of G'.

Let $\{w_1, \ldots, w_s\}$ be the vertex set of H and consider the image $T_1, \ldots, T_s$ of the minor map from H to G'. Let T be a minimal subtree of G' that contains all of the T_i. Such a tree must exist since G' is connected. Let $\hat{H}$ be the graph with the same vertex set as H, and an edge between two vertices w_i, w_j whenever either $w_i w_j \in E(H)$ or when there is a path between T_{w_i} and T_{w_j} in T that is disjoint from any T_{w_k} with $w_i \neq w_k \neq w_j$. We claim that $\hat{H}$ is in the connection closure of H. By construction, $\hat{H}$ is connected and contains all components of H as disjoint subgraphs, so we only need to argue minimality. $\hat{H}$ has no vertices besides those in H so no graph obtained by deleting a vertex would contain all components of H as subgraphs. To see that no edge of $\hat{H}$ is superfluous, we note it has exactly $e + m - 1$ edges and thus no proper subgraph could be connected and have all components of H as disjoint subgraphs. By the construction $\hat{H} \preceq G' \preceq G$, so by the transitivity of the minor relation we have that $\hat{H} \preceq G$.

Conversely let G be an arbitrary graph and assume that $\hat{H} \in \hat{M}$ and $\hat{H} \preceq G$. Because $\hat{H}$ is connected, there is a connected component G' of G such that $\hat{H} \preceq G'$. Now there must be a graph $H \in M(\mathcal{C})$ such that $\hat{H}$ is in the connection closure of H, and since H is a subgraph of $\hat{H}$, $H \preceq \hat{H}$. Then, by the transitivity of the minor relation, $H \preceq G'$ and thus $G' \notin \mathcal{C}$. Therefore G is not a disjoint union of graphs from $\mathcal{C}$. ◀

Now our main theorem is established by a simple induction:

▶ **Theorem 18.** *There is a computable function which takes a set M of excluded minors characterising a minor-closed class $\mathcal{C}$ and $k \geq 0$ to the set $M(\mathcal{C}_k)$.*

Proof. The proof is by induction. For $k = 0$, the set of minimal excluded minors of $\mathcal{C}_0$ is $M(\mathcal{C}_0) = M(\mathcal{C})$, which is given. For $k > 0$, we have that $\mathcal{C}_k = \overline{\mathcal{C}_{k-1}{}^{\text{apex}}}$. By the induction hypothesis we can compute $M(C_{k-1})$, by Theorem 14 we can compute $M(\mathcal{C}_{k-1}{}^{\text{apex}})$ and using Lemma 17 we can compute the connection closure of $M(\mathcal{C}_{k-1}{}^{\text{apex}})$ to obtain $M(\overline{\mathcal{C}_{k-1}{}^{\text{apex}}}) = M(\mathcal{C}_k)$. ◀

So by the Robertson-Seymour Theorem we have the following:

▶ **Corollary 19.** *Let $\mathcal{C}$ be a minor-closed graph class. Then the problem* Elimination Distance to Excluded Minors *is* FPT.

5 Conclusion

We are motivated by the study of the fixed-parameter tractability of edit distances in graphs. Specifically, we are interested in edit distances such as the number of vertex or edge deletions, as well as more involved measures like elimination distance. Aiming at studying general techniques for establishing tractability, we establish an algorithmic meta-theorem showing that any slicewise first-order definable and slicewise nowhere dense problem is FPT. This yields, for instance, the tractability of counting the number of vertex and edge deletions to a class of bounded degree. As a second result, we establish that determining elimination distance to any minor-closed class is FPT, answering an open question of [2].

A natural open question raised by these two results is whether elimination distance to the class of graphs of degree d is FPT. When d is 0, this is just the tree-depth of a graph, and this case is covered by our first result. For positive values of d, it is not clear whether elimination distance is first-order definable. Indeed, a more general version of the question is whether for any nowhere dense and first-order definable $\mathcal{C}$, elimination distance to $\mathcal{C}$ is FPT.

Another interesting case that seems closely related to our methods, but is not an immediate consequence is that of classes that are given by first-order interpretations from nowhere dense classes of graphs. For instance, consider the problem of determining the deletion distance of a graph to a disjoint union of complete graphs. This problem, known as the cluster vertex deletion problem is known to be FPT (see [14]). The class of graphs that are disjoint unions of cliques is first-order definable but certainly not nowhere dense and so the method of Section 3 does not directly apply. However, this class is easily shown to be interpretable in the nowhere dense class of forests of height 1. Can this fact be used to adapt the methods of Section 3 to this class?

References

1 I. Adler, M. Grohe, and S. Kreutzer. Computing excluded minors. In *SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*. SIAM, January 2008.

2 J. Bulian and A. Dawar. Graph isomorphism parameterized by elimination distance to bounded degree. In *Parameterized and Exact Computation – 9th International Symposium, IPEC 2014, Wroclaw, Poland, September 10–12, 2014. Revised Selected Papers*, pages 135–146, 2014.

3 L. Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Inf. Process. Lett.*, 58:171–176, 1996.

4 R. Diestel. *Graph Theory*. Springer, January 2000.

5 R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, October 2012.

6 J. Flum and M. Grohe. Fixed-Parameter Tractability, Definability, and Model-Checking. *SIAM J. Comput.*, 31(1):113–145, 2001.

7 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, May 2006.

8 F. V. Fomin, D. Lokshtanov, N. Misra, and S. 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.

9 J. Gajarský, P. Hlinený, J. Obdrzálek, S. Ordyniak, F. Reidl, P. Rossmanith, F. Sanchez Villaamil, and S. Sikdar. Kernelization using structural parameters on sparse graph classes. In *Algorithms – ESA 2013 – 21st Annual European Symposium*, pages 529–540, 2013.

10 P. A. Golovach. Editing to a Graph of Given Degrees. In *Parameterized and Exact Computation*, pages 196–207. Springer, September 2014.

11 M. Grohe, S. Kreutzer, and S. Siebertz. Deciding first-order properties of nowhere dense graphs. In *STOC'14: Proceedings of the 46th Annual ACM Symposium on Theory of Computing*. ACM, May 2014.

12 J. Guo, F. Hüffner, and R. Niedermeier. A Structural View on Parameterizing Problems: Distance from Triviality. In *Parameterized and Exact Computation*, pages 162–173. Springer, 2004.

13 W. Hodges. *A Shorter Model Theory*. Cambridge University Press, 1997.

14 F. Hüffner, C. Komusiewicz, H. Moser, and R. Niedermeier. Fixed-parameter algorithms for cluster vertex deletion. *Theory Comput. Syst.*, 47:196–217, 2010.

15 D. Marx. Parameterized coloring problems on chordal graphs. *Theor. Comput. Sci.*, 351:407–424, 2006.

16 J. Nesetril and P. Ossona de Mendez. *Sparsity – Graphs, Structures, and Algorithms*. Springer, 2012.

17 R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, February 2006.

18 N. Robertson and P. D. Seymour. Graph minors. XX. wagner's conjecture. *J. Comb. Theory, Ser. B*, 92:325–357, 2004.

Strong ETH and Resolution via Games and the Multiplicity of Strategies

Ilario Bonacina[1] **and Navid Talebanfard**[2]

1 **Computer Science Department, Sapienza University of Rome, via Salaria 113, 00198 Rome, Italy**
bonacina@di.uniroma.it

2 **Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, Ookayama 2-12-1, Meguro-ku, Tokyo 152-8552, Japan**
navid@is.titech.ac.jp

Abstract

We consider a restriction of the Resolution proof system in which at most a fixed number of variables can be resolved more than once along each refutation path. This system lies between regular Resolution, in which no variable can be resolved more than once along any path, and general Resolution where there is no restriction on the number of such variables. We show that when the number of re-resolved variables is not too large, this proof system is consistent with the Strong Exponential Time Hypothesis (SETH). More precisely for large n and k we show that there are unsatisfiable k-CNF formulas which require Resolution refutations of size $2^{(1-\epsilon_k)n}$, where n is the number of variables and $\epsilon_k = \widetilde{O}(k^{-1/5})$, whenever in each refutation path we only allow at most $\widetilde{O}(k^{-1/5})n$ variables to be resolved multiple times. However, these re-resolved variables along different paths do not need to be the same. Prior to this work, the strongest proof system shown to be consistent with SETH was regular Resolution [Beck and Impagliazzo, STOC'13]. This work strengthens that result and gives a different and conceptually simpler game-theoretic proof for the case of regular Resolution.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases Strong Exponential Time Hypothesis, resolution, proof systems

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.248

1 Introduction

The SAT problem is one of the most fundamental NP-complete problems. The theoretical significance of this problem has once again been demonstrated recently, after a series of results showing that SAT if not equivalent to circuit lower bounds but it is at least closely related. Paturi, Pudlák and Zane [19] proved tight depth-3 circuit lower bounds and from their technique they obtained a k-SAT algorithm which beats exhaustive search. Along similar lines, Santhanam [23] modified a lower bound argument to obtain improved satisfiability algorithms for De Morgan formulas of linear size. Employing stronger lower bound arguments, satisfiability algorithms were given for formulas of larger size in [7] and [8]. In a different direction, Williams [27] showed that even small improvements over exhaustive search for satisfiability on certain circuit classes implies a lower bound against that class. In fact he obtained his seminal $\mathsf{NEXP} \not\subseteq \mathsf{ACC}^0$ result in [28] by giving a non-trivial ACC^0-SAT algorithm.

In this paper we will be focusing on the k-SAT problem. There are several non-trivial algorithms known for this problem (see e.g. [11, 19, 18, 24]). Despite this however, the exact

complexity of k-SAT under suitable assumptions remains unknown. Formalising what this complexity could be, Impagliazzo and Paturi [15] formulated the following two hypotheses. The *Exponential Time Hypothesis* (ETH) which states that the are no sub-exponential time algorithms for the SAT problem, and the *Strong Exponential Time Hypothesis* (SETH) which states that the complexity of k-SAT grows as k increases and the running time of the best k-SAT algorithms approach that of exhaustive search. More formally, it says that k-SAT requires running time $2^{(1-\epsilon_k)n}$ where $\epsilon_k \to 0$ as $k \to \infty$.

Both ETH and SETH are stronger than $\mathsf{P} \neq \mathsf{NP}$ and hence we do not expect to be able to verify either of them in any new future. We can however ask whether known algorithms are consistent with these hypotheses, algorithms that work in a relatively intuitive way. For the PPSZ algorithm [18] strong lower bounds were proved in [9] supporting SETH. But one may ask for such a result that holds for a class of algorithms rather than for a specific one. Proof complexity provides a framework to do this. One can think of the run of a SAT algorithm on an unsatisfiable instance as a proof of unsatisfiability. If this proof is structured enough, we can employ tools from proof complexity and obtain lower bounds. For instance practical SAT-solvers are based on the *Davis-Putnam-Logemann-Loveland* algorithm (DPLL) that is a backtracking method introduced by [13, 12] to search for assignments satisfying a CNF formula. It is a well known result that DPLL is equivalent to a sub-system of the proof system Resolution where only proofs having a tree structure are allowed. Hence tree-like Resolution lower bounds transfer to lower bounds for the DPLL algorithm. In a series of works, [16, 25, 17] introduced the idea of *Conflict Driven Clause Learning* (CDCL) as a way for DPLL SAT-solvers to cut the search space and avoid duplicated work. This is done by performing a *conflict analysis* when the search for an assignments leads to a contradiction and then *learning* a clause encoding a reason for that failure. By definition Resolution simulates (polynomially) runs of CDCL solvers over unsatisfiable instances[1], hence lower bounds for Resolution transfer to lower bounds for CDCL solvers.

Exponential lower bounds consistent with ETH have long been known for natural proof systems such as Resolution, see e.g. [26]. These are $2^{\Omega(n)}$ lower bounds for k-CNF formulas on n variables and hence not strong enough to support SETH. Some thirteen years needed to be passed for the first SETH lower bounds. Pudlák and Impagliazzo [22] proved such lower bounds for tree-like Resolution via Prover-Delayer games. Another thirteen years later, Beck and Impagliazzo [4] obtained a very strong width lower bound which simplified and improved the result of [22] for tree-like Resolution and they were able to prove SETH lower bounds for regular Resolution. In this paper we prove another SETH lower bound for regular Resolution. One advantage of our proof is that it gives a SETH lower bound for a proof system which is more general than regular Resolution; we allow at most $\epsilon_k n$ variables to be re-queried along each path. We stress that these re-queried variables along different paths do not need to be the same.

Techniques

A standard technique to prove Resolution size lower bounds is due to Ben-Sasson and Wigderson [5]. They showed that if a formula requires refutations of large width, it also requires refutations with many clauses. More precisely they showed that if a k-CNF formula can only have Resolution refutations of width at least W, then it requires Resolution size at least $2^{(W-k)^2/16n}$, where n is the number of variables. Because of the $\frac{1}{16}$ in the exponent we

[1] The converse also holds *under certain assumptions* on the behaviour of the CDCL solver, cf. [20] and [2].

do not immediately get $2^{(1-\epsilon_k)n}$ size lower bounds from strong width lower bounds. However, we note that if the formula is structured in some sense, for instance if it is a *xorification*, we can avoid this loss.

Beck and Impagliazzo in [4] showed that there are unsatisfiable k-CNF formulas in n variables requiring refutations of size at least $2^{n(1-\epsilon_k)}$ in *regular* Resolution, a sub-system of Resolution. Their proof is an adaptation of a probabilistic technique from [3] and, from an high level can be seen as a variation of the bottleneck counting of Haken in [14]. In their argument a rule is given which maps assignments to particular clauses of the proof, at which a significant amount of 'work' is done.

We will be considering xorification of formulas where we replace each variable with the parity of a block of new variables. For such formulas we can strengthen the result of Ben-Sasson and Wigerson and show that the number of large clauses must be really large, and this gives us the desired lower bound. This result is achieved through *Pudlák games* that characterise Resolution size [21] applied to a structured formula, a xorification of some unsatisfiable CNF φ. This allows us to avoid the use of probabilistic arguments and it is the core of our main technical result, cf. Theorem 4. There we prove that if there is a width lower bound for refuting an unsatisfiable CNF φ in Resolution, then there exists a 'sufficiently strong' exponential size lower bound for refuting a xorification of φ. Our construction apply to a restriction of the Resolution proof system in which at most a fixed number of variables can be resolved more than once along each refutation path. For such system the SETH lower bound for size (Corollary 6) follows for our result on size of xorified formulas (Theorem 4) and from a strong width lower bound for some families of CNFs [4].

Informally, in the *Pudlák game* we have two players, Prover and Delayer, that play on some formula φ. Prover has the objective of showing that the formula φ is unsatisfiable by querying variables. Delayer on the other hand wants to play as long as possible before the formula is falsified while answering to the queries Prover asks her. The size of Resolution proofs of φ is then characterised as the minimal number of *records*, i.e. partial assignments, Prover has to consider in a winning strategy. Hence to prove a Resolution size lower bound we show that, in order to win, Prover must keep a large number of records and we can do that by producing a lot of sufficiently different strategies for Delayer. Prover must win against each of them, hence in his winning strategy he must have a lot of distinct records, since the strategies of Delayer are sufficiently different. In the literature this is done essentially by making Prover play against a Delayer that plays accordingly to a random strategy [21, 10]. Then the size lower bound, that is a lower bound on the number of records that Prover must have in a winning strategy, is obtained by probabilistic arguments. This may very likely lead to some loss in the constants that we need to avoid to prove a SETH lower bound for Resolution size. In the Pudlák game played on the xorification of a formula φ, we give a series of strategies for Delayer to which Prover has to answer in order to win. The construction of strategies relies on the characterisation of Resolution width as a game [1]. At a very high level, a winning strategy for Delayer in the width game on φ gives rise to a multitude of strategies for Delayer on the Pudlák game on the xorification of φ. The new strategies act differently from each other on the xorification of φ, but in a sense they all act the same as the original strategy σ on the original formula φ. This is done by exploiting the combinatorial properties of the xorified formula in such a way that the number of Delayer strategies, for the Pudlák game played on the xorified formula, does indeed hugely amplify. Then, the desired size lower bound follows from a counting argument.

2 Preliminaries

A *literal* is either a variable x or its negation $\neg x$. A *clause* C is a disjunction of literals and by its *width* we mean the number of literals appearing in C and we denote this by $|C|$. A *conjunctive normal form* formula (CNF) is a conjunction of a set of clauses.

Given a boolean function f on a set of variables X, a *partial assignment* is a function $\rho : X \to \{0, 1, *\}$. We call *domain* of ρ, $\mathrm{dom}(\rho)$ the set $\rho^{-1}(\{0, 1\})$. The restriction of f to ρ denoted by $f|_\rho$ is a function on $\rho^{-1}(*)$ obtained from f by fixing the value of all variables in $\rho^{-1}(0) \cup \rho^{-1}(1)$ according to ρ. We write $\rho \subseteq \sigma$ if for all $x \in X$, $\rho(x) \neq *$ implies $\sigma(x) = \rho(x)$. For a partial assignment ρ for which $\rho(x) = *$, by $\rho \cup \{(x, b)\}$ we denote a partial assignment ρ' such that for all $y \neq x$, $\rho'(y) = \rho(y)$ and $\rho'(x) = b$. Given a (partial) assignment ρ and a subset $B \subseteq X$, $\rho|_B$ is a partial assignment defined only on the variables in B such that for all $x \in B$, $\rho|_B(x) = \rho(x)$.

Resolution [6] is a proof system for refuting unsatisfiable CNF formulas. The only inference rule in Resolution is given as follows

$$\frac{C \vee x, \quad D \vee \neg x}{C \vee D},$$

where C and D are clauses and we say that x is *resolved* and $C \vee D$ is called the *resolvant* of $C \vee x$ and $D \vee \neg x$.

A *Resolution derivation* of a clause D from a CNF φ is a sequence $\Pi = \langle C_1, \ldots, C_\tau \rangle$ of clauses such that $C_\tau = D$ and each C_i is either an *axiom*, i.e., a clause from φ, or it is derived by applying the Resolution rule on some clause C_j and $C_{j'}$ such that $j, j' < i$. We will denote this by $\Pi : \varphi \vdash D$. If φ is an unsatisfiable formula, a *Resolution refutation* of φ is a derivation of $\bot$, the empty clause, from φ. Resolution is *sound* and *complete*, that is we can derive $\bot$ from a CNF formula if and only if it is unsatisfiable.

A *δ-regular Resolution derivation* of a clause D from a formula φ in n variables is a Resolution derivation in which along any path at most δn variables are resolved multiple times. Hence a 0-regular Resolution refutation is just a standard regular refutation and a 1-regular Resolution refutation is one without any constraint.

The *size* of a Resolution derivation is the number of clauses appearing in it. We denote the minimum size of a derivation of D from φ by $\mathsf{size}(\varphi \vdash D)$. We also denote the minimum size of a δ-regular derivation of D from φ by $\mathsf{size}_\delta(\varphi \vdash D)$. Similarly we define the *width* of a derivation to be the width of the largest clause appearing in it. We denote the minimum width of a derivation of D from φ by $\mathsf{width}(\varphi \vdash D)$.

3 A game view of Resolution

In this section we present a common framework for the games described by Atserias and Dalmau [1] and Pudlák [21].

▶ **Definition 1** ($\mathsf{Game}(\varphi, \mathcal{R})$)**.** Given an unsatisfiable CNF φ in n variables and a set of partial assignments $\mathcal{R}$ containing the empty assignment, we define a game, $\mathsf{Game}(\varphi, \mathcal{R})$, between two players Prover (he) and Delayer (she).

At each step i of the game a partial assignment $\alpha_i \in \mathcal{R}$ is maintained (α_0 is the empty partial assignment), then at step $i + 1$ the following moves take place:

1. Prover picks some variable $x \notin \mathrm{dom}(\alpha_i)$.
2. Delayer then has to answer $x = b$ for some bit $b \in \{0, 1\}$.
3. Prover set $\alpha_{i+1} \in \mathcal{R}$ such that $\alpha_{i+1} \subseteq \alpha_i \cup \{(x, b)\}$.

If at any point in the game α_i falsify φ then Prover wins; otherwise Delayer wins. We say that Prover has a *winning strategy* for the game if for any strategy of Delayer, he can play so that he wins the game. Otherwise we say that Delayer has a *winning strategy*.

If in each run of the game Prover can query at most δn variables, we call the corresponding game $\mathsf{Game}_\delta(\varphi, \mathcal{R})$.

For a suitable choice of $\mathcal{R}$ the $\mathsf{Game}(\varphi, \mathcal{R})$ is exactly the one used by Atserias and Dalmau [1] to characterise the minimal width of Resolution refutations of φ. In particular in [1] the following result is shown (rephrased here with the notations we just set up).

▶ **Theorem 2** (Atserias and Dalmau [1]). *Let φ be an unsatisfiable CNF and $\mathcal{R}$ be the set of all possible partial assignments with a domain of size strictly less than w. The following are equivalent*

1. Prover *has a winning strategy for* $\mathsf{Game}(\varphi, \mathcal{R})$*;*
2. $\mathsf{width}(\varphi \vdash \bot) < w$.

Due to this equivalence, for this particular choice of $\mathcal{R}$, we will denote $\mathsf{Game}(\varphi, \mathcal{R})$ *by* $\mathsf{width\text{-}Game}(\varphi, w)$.

The next result is essentially due to Pudlák [21]. He shows that we can also characterise the minimal size of Resolution refutations of φ in terms of these games. From a Resolution refutation Π we can construct a winning strategy for Prover with a set $\mathcal{R}$ of the same size of Π and vice versa. Moreover a play of the $\mathsf{Game}_\delta(\varphi, \mathcal{R})$ corresponds to a path in Π and, if Π is δ-regular, in each run the set of variables Prover is going to query many times has size at most δn.

▶ **Theorem 3.** *Let φ be an unsatisfiable CNF and let δ be any real in the interval $[0, 1]$. The following are equivalent*

1. *there exists a set of partial assignments $\mathcal{R}$ such that $|\mathcal{R}| \leq s$ for which* Prover *has a winning strategy for* $\mathsf{Game}_\delta(\varphi, \mathcal{R})$*;*
2. $\mathsf{size}_\delta(\varphi \vdash \bot) \leq s$.

4 Games and Xorifications

Given a CNF φ on the variables $x_1, \ldots, x_n$, we define the *ℓ-xorification* of φ as follows: it is a formula on the new variables y^i_j, where $1 \leq i \leq n$ and $1 \leq j \leq \ell$ and it is obtained by replacing each x_i with $y^i_1 \oplus \ldots \oplus y^i_\ell$. We denote this formula by $\varphi[\oplus^\ell]$ and note that if φ is a k-CNF, then $\varphi[\oplus^\ell]$ can be expanded to a $k\ell$-CNF. Due to this notation we will refer to the variables of φ as the *x-variables* and to the variables of $\varphi[\oplus^\ell]$ as the *y-variables*. Moreover we say that all the y-variables $y^i_1, \ldots, y^i_\ell$ form a *block* of variables corresponding to the x-variable x_i. We say that a partial assignment over the y-variables *fixes* a value for a x-variable x_i if it assigns all the y-variables in the block corresponding to x_i.

▶ **Theorem 4.** *Let φ an unsatisfiable CNF in n variables and w, δ and ℓ be parameters. If* $\mathsf{width}(\varphi \vdash \bot) \geq w$ *then*

$$\mathsf{size}_\delta(\varphi[\oplus^\ell] \vdash \bot) \geq 2^{w\ell(1-\epsilon)},$$

where $\epsilon = \frac{1}{\ell}\log(\frac{e^3\ell n}{w}) + \frac{\delta n}{w}\log\frac{e^3\ell}{\delta}$.

Proof. For each partial assignment α over the y-variables there is naturally associated a partial assignment α' over the x-variables, defined as follows

$$\alpha'(x_i) = \begin{cases} \alpha(y^1_i) \oplus \ldots \oplus \alpha_r(y^\ell_i) & \text{if } \forall j = 1, \ldots, \ell,\ y^j_i \in \mathrm{dom}(\alpha), \\ * & \text{otherwise.} \end{cases}$$

By Theorem 3, it is enough to show that if Prover wins $\mathsf{Game}_\delta(\varphi[\oplus^\ell], \mathcal{R})$ then

$$|\mathcal{R}| \geq 2^{w(\ell - \log(\frac{e^3 \ell n}{w}) - \frac{\delta \ell n}{w} \log \frac{e^3 \ell}{\delta})}.$$

So suppose Prover wins $\mathsf{Game}_\delta(\varphi[\oplus^\ell], \mathcal{R})$ for some set of partial assignments $\mathcal{R}$. Since $\mathsf{width}(\varphi \vdash \bot) \geq w$, by Theorem 2, there is a winning strategy σ for Delayer in the game $\mathsf{width\text{-}Game}(\varphi, w)$.

For each total assignment β on the y-variables, we consider a strategy σ_β for Delayer in the game $\mathsf{Game}_\delta(\varphi[\oplus^\ell], \mathcal{R})$ as follows. Let α_r be the partial assignment on y-variables at stage r of the game $\mathsf{Game}_\delta(\varphi[\oplus^\ell], \mathcal{R})$ and y_j^i the variable queried at stage $r+1$. Then the strategy σ_β for Delayer goes as follows:

1. if there exists $j' \neq j$ such that $y_i^{j'} \notin \mathrm{dom}(\alpha_r)$, set y_i^j to $\beta(y_j^i)$;
2. otherwise, if for all $j' \neq j$, $y_i^{j'} \in \mathrm{dom}(\alpha_r)$, then look at the value $b \in \{0,1\}$ the strategy σ sets the variable x_i when given the partial assignment α'_r. Then set y_i^j to $q \in \{0,1\}$ such that

$$q \oplus \bigoplus_{j' \neq j} \alpha_r(y_i^j) = b.$$

This can be done since $x_i \equiv y_i^1 \oplus \ldots \oplus y_i^\ell$ and the value of x_i can be set freely to 0 or 1 appropriately even after all but one of $y_i^1, \ldots, y_i^\ell$ have been set.

Since we are assuming that Prover has a winning strategy for $\mathsf{Game}_\delta(\varphi[\oplus^\ell], \mathcal{R})$, in particular, this means that for any β he wins against the Delayer's strategy σ_β. It is immediate to see that for each total assignment β over the y-variables, σ_β is a winning strategy for Delayer in the game $\mathsf{width\text{-}Game}(\varphi[\oplus^\ell], w\ell)$. This means that for each total assignment β over the y-variables, $\mathcal{R}$ must contain some partial assignment, denoted by ρ_β, with domain of size at least $w\ell$ and such that at least w blocks of y-variables are completely fixed by ρ_β. Without loss of generality we assume that each ρ_β fixes exactly w blocks of y-variables, that is if ρ_β is setting more y-variables we simply ignore some of the variables and only consider w blocks. Our goal is to show that we have 'many distinct' such partial assignments ρ_β.

Let $B \subseteq [n]$ denote a generic set of size w and consider for each possible such B the set S_B of the total assignments βs such that ρ_β is fixing all the $y_1^i, \ldots, y_\ell^i$ corresponding to some i in B. There are $2^{n\ell}$ possible total assignments β and $\binom{n}{w}$ possible sets B, hence by the pigeonhole principle, there is a set $B^* \subseteq [n]$ of size w such that

$$|S_{B^*}| \geq \frac{2^{n\ell}}{\binom{n}{w}}. \tag{1}$$

Let S'_{B^*} be the set of partial assignments $\beta|_{B^*}$ where $\beta \in S_{B^*}$. We clearly have that

$$|S_{B^*}| \leq |S'_{B^*}| \cdot 2^{n\ell - \ell|B^*|} = |S'_{B^*}| \cdot 2^{n\ell - w\ell}.$$

By equation (1), we get

$$|S'_{B^*}| \geq \frac{2^{w\ell}}{\binom{n}{w}}. \tag{2}$$

We have now that S'_{B^*} and $\{\rho_\beta : \beta \in S_{B^*}\}$ both consist of assignments of domain $\{y_i^j : i \in B^* \wedge 1 \leq j \leq \ell\}$. We show that $|\{\rho_\beta : \beta \in S_{B^*}\}|$ cannot be too small compared to $|S'_{B^*}|$, this will be, intuitively, due to the fact that the βs we start with are very different.

Let Z^β be the set of variables that Prover re-queried when playing against σ_β and for any $i = 1, \ldots, n$ let $Z_i^\beta = Z^\beta \cap \{y_i^1, \ldots, y_i^\ell\}$. By hypothesis $|Z^\beta| \leq \delta\ell n$.

When Delayer follows the strategy σ_β and fixes all y-variables in a block corresponding to x_i, this assignment is within Hamming distance $|Z_i^\beta| + 1$ from β in this block. This means that for each $\beta \in S_{B^*}$ and for each i, $\rho_\beta|_{\{y_1^i,\ldots,y_\ell^i\}}$ has Hamming distance at most $|Z_i^\beta| + 1$ from some partial assignment in S'_{B^*} restricted to $\{y_1^i, \ldots, y_\ell^i\}$. This means that for each $\beta \in S_{B^*}$ and for each i, ρ_β restricted to the set $\{y_1^i, \ldots, y_\ell^i\}$ has Hamming distance at most $|Z_i^\beta| + 1$ from some partial assignment in S'_{B^*} restricted to $\{y_1^i, \ldots, y_\ell^i\}$. Let $\mathcal{Z}$ be the set of all possible sets Z subsets of the y-variables of size $\delta\ell n$ such that there exists $\beta \in S_{B^*}$ with $Z^\beta \subseteq Z$. For any $i = 1, \ldots, n$ let $Z_i = Z \cap \{y_i^1, \ldots, y_i^\ell\}$. Then, by counting the variables where ρ_β and an assignment in S'_{B^*} could differ, we have that

$$|S'_{B^*}| \leq |\{\rho_\beta : \beta \in S_{B^*}\}| \cdot \sum_{Z \in \mathcal{Z}} \prod_{i \in B^*} 2^{|Z_i|+1} \binom{\ell}{|Z_i| + 1}. \tag{3}$$

Hence we have the following chain of inequalities

$$|S'_{B^*}| \overset{eq.(3)}{\leq} |\{\rho_\beta : \beta \in S_{B^*}\}| \cdot \sum_{Z \in \mathcal{Z}} \prod_{i \in B^*} 2^{|Z_i|+1} \binom{\ell}{|Z_i| + 1} \tag{4}$$

$$\leq |\{\rho_\beta : \beta \in S_{B^*}\}| \cdot \sum_{Z \in \mathcal{Z}} \prod_{i \in B^*} \left(\frac{e^2\ell}{|Z_i| + 1}\right)^{|Z_i|+1} \tag{5}$$

$$\leq |\{\rho_\beta : \beta \in S_{B^*}\}| \cdot \sum_{Z \in \mathcal{Z}} \left(\frac{\sum_{i \in B^*} e^2\ell}{\sum_{i \in B^*}(|Z_i| + 1)}\right)^{\sum_{i \in B^*}(|Z_i|+1)} \tag{6}$$

$$\leq |\{\rho_\beta : \beta \in S_{B^*}\}| \cdot \binom{\ell n}{\delta\ell n} \cdot \left(\frac{\sum_{i \in B^*} e^2\ell}{w}\right)^{\delta\ell n+w} \tag{7}$$

$$= |\{\rho_\beta : \beta \in S_{B^*}\}| \cdot \binom{\ell n}{\delta\ell n} \cdot \left(e^2\ell\right)^{\delta\ell n+w} \tag{8}$$

The inequality (6) follows from the weighted AM-GM inequality[2] and the inequality (7) follows from the fact that $w \leq \sum_{i \in B^*}(|Z_i| + 1) \leq \delta\ell n + w$. Putting all together we have that

$$\begin{aligned} |\mathcal{R}| \overset{(\dagger\dagger)}{\geq} |\{\rho_\beta : \beta \in S_{B^*}\}| \geq \frac{|S'_{B^*}|}{\binom{n\ell}{\delta\ell n}\left(e^2\ell\right)^{\delta\ell n+w}} &\overset{\text{(eq. 2)}}{\geq} \frac{2^{w\ell}}{\binom{n}{w}\binom{\ell n}{\delta\ell n}\left(e^2\ell\right)^{\delta\ell n+w}} \\ &\geq \frac{2^{w\ell}}{\left(\frac{en}{w}\right)^w \left(\frac{e}{\delta}\right)^{\delta\ell n} \left(e^2\ell\right)^{\delta\ell n+w}} \\ &= 2^{w(\ell - \log(\frac{e^3\ell n}{w}) - \frac{\delta\ell n}{w}\log\frac{e^3\ell}{\delta})}, \end{aligned}$$

where the inequality (††) follows by the definition of ρ_β. ◀

The next step now is to obtain formulas which require very large Resolution width. Such a construction is given by Beck and Impagliazzo in [4].

---

[2] The *weighted Arithmetic Mean - Geometric Mean inequality* says that given non-negative numbers $a_1, \ldots, a_n$ and non-negative weights $w_1, \ldots, w_n$ then

$$\prod_i a_i^{w_i} \leq \left(\frac{\sum_i w_i a_i}{w}\right)^w,$$

where $w = \sum_i w_i$. We applied this inequality with $a_i = e^2\ell$ and $w_i = |Z_i| + 1$.

▶ **Theorem 5** ([4]). *For any large n and k, there exist an unsatisfiable k-CNF formula φ on n variables and some $\zeta_k = \widetilde{O}(k^{-1/4})$ such that*

$$\mathsf{width}(\varphi \vdash \bot) \geq (1-\zeta_k)n.$$

Now, informally, our SETH lower bound for Resolution will follow from the existence of a CNF requiring very high Resolution width (Theorem 5) and the previous theorem about xorifications (Theorem 4).

▶ **Corollary 6.** *For any large n, k and $\ell = \widetilde{\Theta}(k^{1/4})$, there exists an unsatisfiable k-CNF formula φ on n variables such that*

$$\mathsf{size}_\delta(\varphi[\oplus^\ell] \vdash \bot) \geq 2^{(1-\epsilon_{k'})n\ell},$$

where $k' = k\ell$ is the initial width of the clauses of $\varphi[\oplus^\ell]$ and $\epsilon_{k'} = \delta = \widetilde{O}(k'^{-1/5})$.

Proof. Let φ be the k-CNF formula given by Theorem 5, in particular $\mathsf{width}(\varphi \vdash \bot) \geq (1-\zeta_k)n$ where $\zeta_k = \widetilde{O}(k^{-1/4})$. Then $\varphi[\oplus^\ell]$ is a k'-CNF on $n\ell$ variables where $k' = k\ell$. By the choice of $\ell = \widetilde{\Theta}(k^{1/4})$, $\delta = \widetilde{O}(k^{-1/4})$ and by Theorem 4, it follows that

$$\begin{aligned}\mathsf{size}_\delta(\varphi[\oplus^\ell] \vdash \bot) &\geq 2^{(1-\zeta_k)n(\ell - \log(\frac{e^3\ell n}{w}) - \frac{\delta \ell n}{w}\log\frac{e^3\ell}{\delta})} \\ &\overset{(\dagger)}{=} 2^{(1-\zeta_k)n(\ell - O(\log k) - \ell\widetilde{O}(k^{-1/4}))} = 2^{(1-\widetilde{O}(k^{-1/4}))n\ell} \\ &= 2^{(1-\epsilon_{k'})n\ell}.\end{aligned}$$

In particular the equality (†) follows from the choice of $\ell = \widetilde{\Theta}(k^{1/4})$ and $\delta = \widetilde{O}(k^{-1/4})$. To obtain the asymptotic behaviour of $\epsilon_{k'}$ with respect to k', just observe that $k' = k\ell = \widetilde{\Theta}(k^{5/4})$ and $\epsilon_{k'} = \widetilde{O}(k^{-1/4})$, hence $\epsilon_{k'} = \widetilde{O}(k'^{-1/5})$. Similarly we get the asymptotic behaviour of δ as a function of k'. ◀

5 Conclusion

We proved that there exist unsatisfiable k-CNF formulas in n variables that require δ-regular Resolution refutations of size at least $2^{(1-\epsilon)n}$, where $k = \widetilde{O}(\epsilon^{-5})$ and where $\delta = \widetilde{O}(\epsilon^{-5})$. A natural question is whether it is possible to improve the dependency of δ and k on ϵ.

More generally, we have some proof systems stronger than Resolution, such as Polynomial Calculus + Resolution, RES(k), Cutting Planes, for which we know that there are some unsatisfiable CNFs which require exponential size refutations. Are those proof systems consistent with SETH?

Acknowledgments. We would like to thank Nicola Galesi for discussions on the topic. We would also like to thank Jakob Nordström and Massimo Lauria for discussions on Resolution size and strong width lower bounds.

References

1 Albert Atserias and Víctor Dalmau. A combinatorial characterization of resolution width. *J. Comput. Syst. Sci.*, 74(3):323–334, 2008.

2 Albert Atserias, Johannes Klaus Fichte, and Marc Thurley. Clause-learning algorithms with many restarts and bounded-width resolution. *J. Artif. Intell. Res. (JAIR)*, 40:353–373, 2011.

3 Paul Beame, Christopher Beck, and Russell Impagliazzo. Time-space tradeoffs in resolution: superpolynomial lower bounds for superlinear space. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19–22, 2012*, pages 213–232. ACM, 2012.

4 Christopher Beck and Russell Impagliazzo. Strong ETH Holds for Regular Resolution. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC'13, pages 487–494. ACM, 2013.

5 Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow – resolution made simple. *J. ACM*, 48(2):149–169, 2001.

6 Archie Blake. *Canonical Expressions in Boolean Algebra*. PhD thesis, University of Chicago, 1937.

7 Ruiwen Chen, Valentine Kabanets, Antonina Kolokolova, Ronen Shaltiel, and David Zuckerman. Mining circuit lower bound proofs for meta-algorithms. In *IEEE 29th Conference on Computational Complexity, CCC*, pages 262–273, 2014.

8 Ruiwen Chen, Valentine Kabanets, and Nitin Saurabh. An improved deterministic #SAT algorithm for small de morgan formulas. In *Mathematical Foundations of Computer Science 2014 – 39th International Symposium, MFCS*, pages 165–176, 2014.

9 Shiteng Chen, Dominik Scheder, Navid Talebanfard, and Bangsheng Tang. Exponential Lower Bounds for the PPSZ k-SAT Algorithm. In *SODA*, pages 1253–1263, 2013.

10 Stefan S. Dantchev. Relativisation provides natural separations for resolution-based proof systems. In *Computer Science – Theory and Applications, First International Computer Science Symposium in Russia, CSR 2006, St. Petersburg, Russia, June 8-12, 2006, Proceedings*, pages 147–158, 2006.

11 Evgeny Dantsin, Andreas Goerdt, Edward A. Hirsch, Ravi Kannan, Jon M. Kleinberg, Christos H. Papadimitriou, Prabhakar Raghavan, and Uwe Schöning. A deterministic $(2-2/(k+1))^n$ algorithm for k-SAT based on local search. *Theor. Comput. Sci.*, 289(1):69–83, 2002.

12 Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.

13 Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, 1960.

14 Armin Haken. The intractability of resolution. *Theor. Comput. Sci.*, 39:297–308, 1985.

15 Russell Impagliazzo and Ramamohan Paturi. On the Complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

16 Roberto J. Bayardo Jr. and Robert Schrag. Using CSP look-back techniques to solve real-world SAT instances. In Benjamin Kuipers and Bonnie L. Webber, editors, *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 97, IAAI 97, July 27-31, 1997, Providence, Rhode Island.*, pages 203–208. AAAI Press / The MIT Press, 1997.

17 Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*, pages 530–535. ACM, 2001.

18 Ramamohan Paturi, Pavel Pudlák, Michael E. Saks, and Francis Zane. An improved exponential-time algorithm for k-SAT. *J. ACM*, 52(3):337–364, 2005.

19 Ramamohan Paturi, Pavel Pudlák, and Francis Zane. Satisfiability coding lemma. In *38th Annual Symposium on Foundations of Computer Science, FOCS*, pages 566–574, 1997.

20 Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning SAT solvers as resolution engines. *Artif. Intell.*, 175(2):512–525, 2011.

21 Pavel Pudlák. Proofs as games. *American Mathematical Monthly*, 107(6):541–550, 2000.

22 Pavel Pudlák and Russell Impagliazzo. A lower bound for DLL algorithms for k-SAT (preliminary version). In *SODA*, pages 128–136, 2000.

23 Rahul Santhanam. Fighting perebor: New and improved algorithms for formula and QBF satisfiability. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010*, pages 183–192, 2010.

24 Uwe Schöning. A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *40th Annual Symposium on Foundations of Computer Science, FOCS,*, pages 410–414, 1999.

25 João P. Marques Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. Computers*, 48(5):506–521, 1999.

26 Alasdair Urquhart. Hard examples for resolution. *J. ACM*, 34(1):209–219, 1987.

27 Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC*, pages 231–240, 2010.

28 Ryan Williams. Non-uniform ACC circuit lower bounds. In *Proceedings of the 26th Annual IEEE Conference on Computational Complexity, CCC*, pages 115–125, 2011.

Quick but Odd Growth of Cacti*

Sudeshna Kolay[1], Daniel Lokshtanov[2], Fahad Panolan[1], and Saket Saurabh[1,2]

1 **Institute of Mathematical Sciences, Chennai, India**
2 **University of Bergen, Norway**

Abstract

Let $\mathcal{F}$ be a family of graphs. Given an input graph G and a positive integer k, testing whether G has a k-sized subset of vertices S, such that $G \setminus S$ belongs to $\mathcal{F}$, is a prototype vertex deletion problem. These type of problems have attracted a lot of attention in recent times in the domain of parameterized complexity. In this paper, we study two such problems; when $\mathcal{F}$ is either a family of cactus graphs or a family of odd-cactus graphs. A graph H is called a *cactus* graph if every pair of cycles in H intersect on at most one vertex. Furthermore, a cactus graph H is called an *odd cactus*, if every cycle of H is of odd length. Let us denote by $\mathcal{C}$ and $\mathcal{C}_{\mathsf{odd}}$, families of cactus and odd cactus, respectively. The vertex deletion problems corresponding to $\mathcal{C}$ and $\mathcal{C}_{\mathsf{odd}}$ are called Diamond Hitting Set and Even Cycle Transversal, respectively. In this paper we design randomized algorithms with running time $12^k n^{\mathcal{O}(1)}$ for both these problems. Our algorithms considerably improve the running time for Diamond Hitting Set and Even Cycle Transversal, compared to what is known about them.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases Even Cycle Transversal, Diamond Hitting Set, Randomized Algorithms

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.258

1 Introduction

In the field of parameterized graph algorithms, vertex (edge) deletion (addition, editing) problems constitute a considerable fraction. In particular, let $\mathcal{F}$ be a family of graphs. Given an input graph G and a positive integer k, testing whether G has a k-sized subset of vertices (edges) S, such that $G - S$ belongs to $\mathcal{F}$, is a prototype vertex (edge) deletion problem. Many well known problems in parameterized complexity can be phrased in this language. For example, if $\mathcal{F}$ is a family of edgeless graphs, or forests or bipartite graphs, then it corresponds to Vertex Cover, Feedback Vertex Set, and Odd Cycle Transversal, respectively. Most of these problems are NP-complete due to a classic result by Lewis and Yannakakis [13], and naturally a candidate for parameterized study (with respect to solution size). Vertex Cover, Feedback Vertex Set and Odd Cycle Transversal are some of the most well studied problem in the domain of parameterized complexity. These problems have led to identification of several new techniques and ideas in the field.

Recent years have seen a plethora of results around vertex and edge deletion problems, in the domain of parameterized complexity [3, 4, 8, 9, 10, 11, 12]. In this paper, we continue this line of research and study two vertex deletion problems. In particular we study the

* The research leading to these results has received funding from the European Research Council (ERC) via grants Rigorous Theory of Preprocessing, reference 267959 and PARAPPROX, reference 306992 and the Bergen Research Foundation via grant "Beating Hardness by Preprocessing".

problem of deleting vertices to get a cactus or an odd cactus graph. A graph H is called a *cactus* graph if every pair of cycles in H intersect on at most one vertex. Furthermore, a cactus graph H is called an *odd cactus* graph, if every cycle of H is of odd length. Let us denote by $\mathcal{C}$ and $\mathcal{C}_{\mathsf{odd}}$, families of cacti and odd cacti, respectively. The vertex deletion problems corresponding to $\mathcal{C}$ and $\mathcal{C}_{\mathsf{odd}}$ are called Diamond Hitting Set and Even Cycle Transversal, respectively. It is important to note here that the name of deleting vertices to get into $\mathcal{C}_{\mathsf{odd}}$ is called Even Cycle Transversal, because it is equivalent to deleting a k-sized subset S such that $G - S$ does not have any *cycle of even length*. More precisely, we study the following problems:

Even Cycle Transversal	**Parameter:** k
Input: An undirected graph G and a positive integer k.	
Question: Does there exist a set S such that $G - S \in \mathcal{C}_{\mathsf{odd}}$?	

Diamond Hitting Set	**Parameter:** k
Input: An undirected graph G and a positive integer k.	
Question: Does there exist a set S such that $G - S \in \mathcal{C}$?	

It needs to be mentioned that, in this paper, we refer to multigraphs (may have parallel edges) as graphs. While Odd Cycle Transversal is one of the most well studied problem in the realm of parameterized complexity, there is only one article about Even Cycle Transversal in the literature. The structure of the graph without even cycles, or without cycles 0 modulo some positive integer p, is simple. Thomassen showed that such graphs have treewidth at most $f(p)$ [16]. Misra et al. [15] used the structural properties of an odd-cactus graph to design an algorithm for Even Cycle Transversal with running time $50^k n^{\mathcal{O}(1)}$. They also give an $\mathcal{O}(k^2)$ kernel for the problem. On the other hand the family of cacti $\mathcal{C}$ can be characterised by a single excluded minor. In particular, let Θ be a graph on two vertices that have three parallel edges, then a graph $H \in \mathcal{C}$ if and only if H does not contain Θ as a minor. Since Θ is a connected planar graph we obtain a $c^k n^{\mathcal{O}(1)}$ time algorithm as a corollary to the main results in [8, 11, 12]. It also has $\mathcal{O}(k^2)$ kernel [7]. However, we are not aware of exact value of c as all these algorithms use a protrusion subroutine [2]. In this paper we give the following algorithm for these problems.

▶ **Theorem 1.** *There is a randomised algorithm for* Diamond Hitting Set *and* Even Cycle Transversal *running in time* $12^k n^{\mathcal{O}(1)}$.

Our Methods. Our algorithms use the same methodology that is used for the $4^k n^{\mathcal{O}(1)}$ time algorithm for Feedback Vertex Set [1], and its generalization to Planar $\mathcal{F}$ Deletion [8]. In both our algorithms, we start by applying some reduction rules to the given instance. After this, we show that the number of edges incident to any solution S of our problems, is a constant fraction to the total number of edges in the graph. This counting lemma is our main technical contribution. We also observe that the analysis for the counting lemma is tight for an infinite family of graphs and thus the analysis of our randomized algorithms can not be improved. It is in the same spirit as finding an infinite family of instances for which an approximation algorithm achieves its approximation ratio.

To apply our reduction rules in a way that this fraction is as small as possible, we study a more general problem than Even Cycle Transversal, which we call Parity Even Cycle Transversal. In this problem we are given a graph G and a weight function $w : E(G) \to \{0, 1\}$ and the objective is to delete a subset S of vertices of size at most k such

that in $G - S$ there is no cycle whose weight sum is even. Observe that if w assigns one to every edge then it is same as Even Cycle Transversal. We conclude the introduction by noting that Diamond Hitting Set and Even Cycle Transversal admit approximation algorithms with factor 9 and 10 respectively [6, 15].

2 Preliminaries

We denote a graph as G, while its vertex set and edge set as $V(G)$ and $E(G)$ respectively. It is possible that there are parallel edges between two vertices of a graph. The degree of a vertex $v \in V(G)$, denoted by $d_G(v)$, is the number of edges incident on v. The neighbourhood of v, denoted by $N_G(v)$, is the set of vertices that have at least one edge with v. $N^2_G(v)$ is the set of vertices that have a path of length at most two with v. For a subset of vertices S, the subgraph of G induced by S is denoted by $G[S]$. Similarly, for a subset of edges E', the subgraph of G induced by E' is denoted by $G[E']$. For $S \subseteq V(G)$, $G - S$ denotes the induced subgraph $G[V(G) - S]$. Similarly, for $E' \subseteq E(G)$, $G - E'$ denotes the induced subgraph $G[E(G) - E']$. An edge between two vertices $u, v \in V(G)$ is denoted by (u, v), while a path between u, v is denoted by $[u, v]$. If a sequence of vertices $v_1, \ldots, v_t$ or edges $e_1, \ldots e_t$ form a path, then too we denote this path by $[v_1, \ldots, v_t]$ and $[e_1, \ldots e_t]$ respectively. Given two subsets $V_1, V_2 \subseteq V(G)$, $E(V_1, V_2)$ denotes the set of edges in $E(G)$ that have one end point in V_1 and the other in V_2. The subdivision of an edge $e = (u, v)$ of a graph G results in a graph G', which contains a new vertex w, and where the edge e is replaced by two new edges (u, w) and (w, v). A graph $\hat{G}$ is a subdivision of a graph G if there is a sequence of graphs $\{G_1, G_2, \ldots, G_t\}$, with $G_1 = G$ and $G_t = \hat{G}$, where for each $1 < i \leq t$, G_i is obtained by the subdivision of an edge of G_{i-1}.

▶ **Definition 2.** Given a graph G, a cut vertex of G is a vertex v such that $G - \{v\}$ has more components than G. A block of G is a maximal connected subgraph that does not contain any cut vertices of G. A block-decomposition of G is the collection of all blocks. It corresponds to a tree $\mathcal{T}$, where a block X of G corresponds to a vertex t_X of $\mathcal{T}$, and $(t_X, t_Y) \in E(\mathcal{T})$ if the intersection of the corresponding blocks X, Y is exactly one cut vertex.

A block decomposition of a graph can be built in polynomial time.

▶ **Lemma 3** (†). [1] *Let T be a tree. Let $V_1 = \{v \in V(T) \mid d_T(v) = 1\}$, $V_2 = \{v \in V(T) \mid d_T(v) = 2\}$ and $V_3 = \{v \in V(T) \mid d_T(v) \geq 3\}$. Then $\sum_{v \in V_3} d_T(v) \leq 3|V_1|$.*

▶ **Definition 4.** A cactus graph is a connected graph where any two cycles have at most one vertex in common. Equivalently, every edge of the graph belongs to at most one cycle. Another equivalent definition is that a block of a cactus graph can be either a cycle or an edge. A graph where every component is a cactus graph is called a forest of cacti.

▶ **Definition 5.** Let H be a graph on a pair of vertices $\{u, v\}$ that have 3 parallel edges between them. A graph is called a diamond graph if it is obtained by a number of subdivisions of H.

The following Proposition characterizes the class of forests of cacti.

▶ **Proposition 6.** *A graph is a forest of cacti if and only if it does not have a diamond as a subgraph.*

[1] Results marked with † can be found in the full version.

The definition of diamond graphs and the characterisation of forests of cacti have been taken from [6]. Please refer to [5] for further details on notations and definitions in Graph Theory.

3 Counting Lemma

In this section, we consider a graph G which has a set S, the deletion of which results in a cactus graph. Moreover, each vertex of the cactus graph has at least three distinct neighbors in G or shares at least two edges with S. Then, it is possible to bound the number of edges in $E(G-S)$ by the number of edges in $E(S, V(G) \setminus S)$. In fact, we exhibit a family of graphs where this bound is tight, up to a constant difference.

▶ **Lemma 7.** *Let G be a graph and $S \subseteq V(G)$ such that $G - S$ is a cactus graph and for all $v \in V(G) \setminus S$ one of the following two conditions holds:*
1. *v has at least 3 distinct neighbors in G, or*
2. *there are at least two edges in $E(v, S)$*
Then $|E(G - S)| \leq 5|E(S, V(G) \setminus S)|$.

Proof. Let $G' = G - S$. We know that G' is a cactus graph. Let $\mathcal{T}$ be the block decomposition tree of G' rooted at a vertex of degree one. Let $B = E(G')$ and $C = E(S, V(G) \setminus S)$ We need to show that $|B| \leq 5|C|$.

Towards the proof, we first define some notations. Let X is a block of size at most 2 (an edge or a cycle of length 2) in G' such that t_X has only one child, which is a leaf node in $\mathcal{T}$. Then we say X and Y together form a *super block*. If blocks X and Y form a super block Z, where t_Y is a leaf node, then by parent of the super block Z, we mean the parent of t_X in $\mathcal{T}$. All other blocks, which are not part of any super block, are called a *normal blocks*. By *size* of a (super/normal) block Z, denoted by $\mathsf{size}(Z)$, we mean the number of edges in the block Z. To bound the number of edges in G' it is enough to bound the total number of edges in super blocks and normal blocks. Let $\mathcal{B}_\ell$ be the set containing all super blocks and normal blocks which correspond to leaves in $\mathcal{T}$. Let $\mathcal{B}_n$ be the set of normal blocks which are not part of $\mathcal{B}_\ell$. Now we define B_ℓ as the set of edges in the (normal/super) blocks which are part of $\mathcal{B}_\ell$, and B_n as the set of edges in the normal blocks which are part of $\mathcal{B}_n$. To bound the cardinality of B, it is enough to bound the cardinality of B_ℓ and B_n, individually. We partition the edges in C as follows. We say an edge $e \in C$ is incident to a (super/normal) block Z if it is incident to a vertex u in Z, which is not the cut vertex shared with the parent of Z. We use E_Z to denote the set of edges in C, which are incident to the (super/normal) block Z. Let C_ℓ be the set of edges in C which are incident to (super/normal) blocks in $\mathcal{B}_\ell$. Similarly, let C_n be the set of edges in C which are incident to blocks in $\mathcal{B}_n$. Let r_i be the number of blocks of size i in $\mathcal{B}_\ell$. Let $B_\ell^{(i)}$ be the set of edges in blocks of size i in $\mathcal{B}_\ell$. Let $C_\ell^{(i)}$ be the set of edges in C_ℓ which are incident to blocks of size i in $\mathcal{B}_\ell$. Notice that $B_\ell = \biguplus_i B_\ell^{(i)}$ and $C_\ell = \biguplus_i C_\ell^{(i)}$.

▶ **Claim 1.** *$r_i \leq \frac{|C_\ell^{(i)}|}{2}$ for $i \leq 4$ and $r_i \leq \frac{|C_\ell^{(i)}|}{i-3}$ for $i \geq 5$.*

Proof.

Bound on r_1. Let X be a block of size one in $\mathcal{B}_\ell$. That is, the block X is a single edge (x, y) and there is a vertex in $\{x, y\}$ which has degree one in G'. Let x be the degree one vertex. By our assumption at least 2 edges in $C_\ell^{(1)}$ are incident on x. This implies that $|E_X| \geq 2$. Thus we have that $|C_\ell^{(1)}| = \sum_{\{X:\mathsf{size}(X)=1\}} E_X \geq 2r_1$. Hence $r_1 \leq \frac{|C_\ell^{(1)}|}{2}$.

Bound on r_2. Let X be a block of size two in $\mathcal{B}_\ell$. If X is a normal block, then the block X is a cycle y, x, y of length 2. Since X is leaf block, there is a vertex in X which is not a cut vertex in G'. Let x be the vertex in X such that x is not a cut vertex. This implies that $N_{G'}(x) = \{y\}$. Thus, by our assumption, either $|E(x, S)| \geq 2$ or x has two neighbors in S. In either case, $|E(x, S)| \geq 2$. That is, $|E_X| \geq 2$. If X is a super block, then X consists of two blocks Y and Z of size 1 each, such that t_Y has only one child t_Z and t_Z is a leaf node in $\mathcal{T}$. Let $Z = (x, y)$ be such that x has degree one in G'. Thus, by our assumption, we can conclude that $|E(x, S)| \geq 2$. That is, $|E_X| \geq 2$. Thus, we have that $|C_\ell^{(2)}| = \sum_{\{X:\mathsf{size}(X)=2\}} E_X \geq 2r_2$. Hence, $r_2 \leq \frac{|C_\ell^{(2)}|}{2}$.

Bound on r_3. Let X be a (super/normal) block of size three in $\mathcal{B}_\ell$. That is, either the block X is a cycle x, y, z, x of length 3, or it is a super block consisting of two blocks, where one of them is a cycle of length 2 and other is an edge. If X is a cycle x, y, z, x, then t_X is a leaf in $\mathcal{T}$. Let z be the only cut vertex in $\{x, y, z\}$. This implies that the degrees of x and y are exactly 2 in G'. Thus, by our assumption, $|E(x, S)| \geq 1$ and $|E(y, S)| \geq 1$. This implies that $|E_X| \geq 2$.

Suppose X is a super block. Then X consists of a cycle x, y, x and an edge (y, z). In this case, only one vertex, either x or z, will be shared with the parent of X and all other vertex will not have a neighbor in $G' - X$. Suppose x is the shared vertex with the parent of the block X. Then the number of distinct neighbors of y and z are exactly 2 and 1 respectively in G'. This implies that $|E(y, S)| \geq 1$ and $|E(z, S)| \geq 2$. Consequently, $|E_X| \geq 3$. By a similar argument, we can show that if z is the shared vertex of the super block X with its parent, then $|E_X| \geq 3$. Thus, we have that $|C_\ell^{(3)}| = \sum_{\{X:\mathsf{size}(X)=3\}} E_X \geq 2r_3$. Hence, $r_3 \leq \frac{|C_\ell^{(3)}|}{2}$.

Bound on r_4. Let X be a (super/normal) block of size four in $\mathcal{B}_\ell$. That is, either the block X is a cycle of length 4 or it is a super block consisting of two blocks. If X is a cycle of length 4, then t_X is a leaf in $\mathcal{T}$. This implies that the degree of every vertex in X, except the cut vertex shared with the parent block, is exactly 2 in G'. This implies that $|E_X| \geq 3$.

Suppose X is a super block consisting of two blocks Y and Z, where size of Y is at most 2 and t_Z is a leaf node in $\mathcal{T}$. If $\mathsf{size}(Y) = 1$, then Z is a cycle of length 3. This implies that at least two vertices in Z has degree exactly 2 in G'. Thus, by our assumption, $|E_Z| \geq 2$ and this implies that $|E_X| \geq 2$.

If $\mathsf{size}(Y) = 2$, then both Y and Z are cycles of length 2. Let x, y, x be the block Y and y, z, y be the block Z. Thus, the number of distinct neighbors of y and z in G' is 2 and 1 respectively. By our assumption, this implies that $|E(y, S)| \geq 1$ and $|E(z, S)| \geq 2$. Thus, we have that $|E_X| \geq 3$. Hence, we conclude that $|C_\ell^{(4)}| = \sum_{\{X:\mathsf{size}(X)=4\}} E_X \geq 2r_4$. This means, $r_4 \leq \frac{|C_\ell^{(4)}|}{2}$.

Bound of r_i for $i \geq 5$. Let X be a (super/normal) block of size at least five in $\mathcal{B}_\ell$. That is, either the block X is a cycle of length i, or it is a super block consisting of two blocks Y and Z such that Z is a cycle of length at least $i - 2$ and t_Z is a leaf in $\mathcal{T}$. In either case, X contains at least $i - 3$ vertices (excluding the cut vertex shared with the parent block) having exactly 2 distinct neighbors in G'. This implies that $|E_X| \geq i - 3$. Hence, we have that $|C_\ell^{(i)}| = \sum_{\{X:\mathsf{size}(X)=i\}} E_X \geq (i - 3)r_i$. Thus, $r_i \leq \frac{|C_\ell^{(i)}|}{i-3}$. ◀

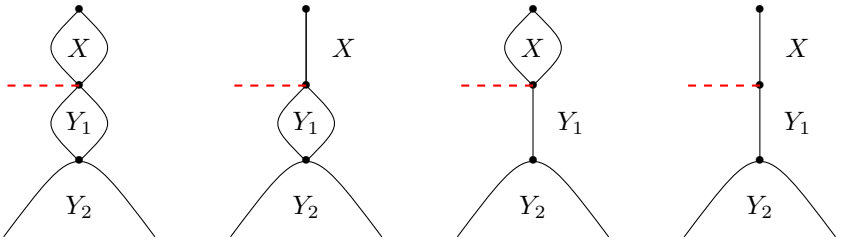


Figure 1 A schematic diagram, when a block X of size at most 2 has only one child which is a super block composed of Y_1 and Y_2. Here the red colored dotted edges belongs to $E(S, V(G) \setminus S)$.

Now we can bound the cardinality of B_ℓ. Let $C_\ell^{(\leq 4)} = \bigcup_{i \leq 4} C_\ell^{(i)}$ and $C_\ell^{(\geq 5)} = \bigcup_{i \geq 5} C_\ell^{(i)}$.

$$|B_\ell| = \sum_i |B_\ell^{(i)}| = \sum_i i \cdot r_i \tag{1}$$

$$\leq 2|C_\ell^{(\leq 4)}| + \sum_{i \geq 5} \frac{i}{i-3} |C_\ell^{(i)}| \qquad \text{(By Claim 1)}$$

$$\leq 2|C_\ell^{(\leq 4)}| + \frac{5}{2}|C_\ell^{(\geq 5)}| \tag{2}$$

What remains is to bound the cardinality of B_n. Let $\mathcal{B}_n^{(\geq 3)}$ be the set of blocks in $\mathcal{B}_n$ such that the corresponding nodes in $\mathcal{T}$ have degree at least 3. That is,

$$\mathcal{B}_n^{(\geq 3)} = \{X \in \mathcal{B}_n \mid d_{\mathcal{T}}(t_X) \geq 3\}.$$

Let $B_n^{(\geq 3)}$ be the set of edges present in the blocks in $\mathcal{B}_n^{(\geq 3)}$. We first bound the cardinality of $B_n^{(\geq 3)}$ and then the cardinality of $B_n \setminus B_n^{(\geq 3)}$. For a set $X \subseteq V(G')$ let numcut_X and $\mathsf{numnoncut}_X$ denote the number of cut vertices and non-cut vertices in X, respectively.

$$|B_n^{(\geq 3)}| = \sum_{X \in \mathcal{B}_n^{(\geq 3)}} |X|$$

$$= \sum_{X \in \mathcal{B}_n^{(\geq 3)}} \mathsf{numcut}_X + \mathsf{numnoncut}_X \tag{3}$$

The quantity $\sum_{X \in \mathcal{B}_n^{(\geq 3)}} \mathsf{numcut}_X$, is at most $\sum_{X \in \mathcal{B}_n^{(\geq 3)}} d_{\mathcal{T}}(t_X)$. This is bounded by three times the number of leaves in $\mathcal{T}$ (by Lemma 3). Thus by Claim 1,

$$\sum_{X \in \mathcal{B}_n^{(\geq 3)}} \mathsf{numcut}_X \leq \frac{3}{2}|C_\ell^{(\leq 4)}| + \frac{3}{2}|C_\ell^{(\geq 5)}| \tag{4}$$

Let $C_n^{\geq 3}$ be the set of edges in C_n which are incident to blocks in $\mathcal{B}_n^{(\geq 3)}$, and $C_n^{\leq 2}$ be the set of edges in C_n which are incident to blocks in $\mathcal{B}_n \setminus \mathcal{B}_n^{(\geq 3)}$. For each non-cut vertex x in the block $X \in \mathcal{B}_n^{(\geq 3)}$, there is at least one edge from $C_n^{(\geq 3)}$ which is incident on x. This implies that

$$\sum_{X \in \mathcal{B}_n^{(\geq 3)}} \mathsf{numnoncut}_X \leq |C_n^{(\geq 3)}| \tag{5}$$

Applying Equations 4 and 5 in Equation 3, we get that

$$|B_n^{(\geq 3)}| \leq \frac{3}{2}|C_\ell^{(\leq 4)}| + \frac{3}{2}|C_\ell^{(\geq 5)}| + |C_n^{(\geq 3)}| \tag{6}$$

Now we bound the cardinality of $B_n \setminus B_n^{(\geq 3)}$. First, we bound the number of edges in the blocks in $\mathcal{B}_n \setminus \mathcal{B}_n^{(\geq 3)}$ which are not incident to any edge in C_n. Let X be a block in $\mathcal{B}_n \setminus \mathcal{B}_n^{(\geq 3)}$,

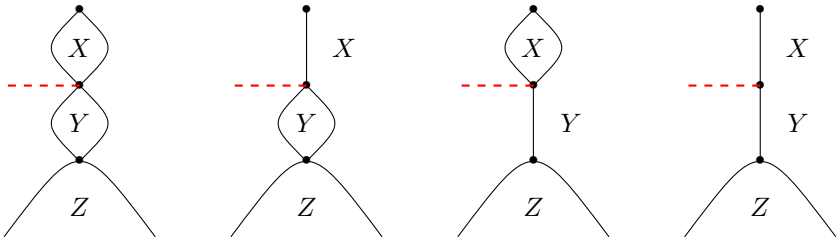


Figure 2 A schematic diagram, when a block X of size at most 2 has only one child Y such that $size(Y) \leq 2$ and $d_{\mathcal{T}}(t_Y) = 2$. Here the red colored dotted edges belongs to $E(S, V(G) \setminus S)$.

such that there is no edge from C_n incident on it. Since t_X has degree 2 in $\mathcal{T}$, the number of cut vertices in X is 2. Now, we claim that $\mathsf{size}(X) \leq 2$. Suppose not. Then there is a vertex x in X such that the degree of x in G' is two. Thus, by our assumption, x is incident to an edge from C_n. This contradicts the fact that there is no edge from C_n is incident on X. Since X is a block in $\mathcal{B}_n \setminus \mathcal{B}_n^{(\geq 3)}$, we have that t_X has only one child. Let the child of t_X be t_Y. Now we have the following claim.

▶ **Claim 2.** *Either $d_{\mathcal{T}}(t_Y) \geq 3$ or $Y \in \mathcal{B}_n \setminus \mathcal{B}_n^{(\leq 3)}$ such that there is an edge from $C_n^{(\leq 2)}$ incident on Y.*

Proof. Towards the claim, we first show that $Y \notin \mathcal{B}_\ell$. Suppose not. If Y is a normal block in $\mathcal{B}_\ell$, then X and Y together will form a super block and it contradicts the fact that $X \in \mathcal{B}_n \setminus \mathcal{B}_n^{(\geq 3)}$. Suppose Y is a super block in $\mathcal{B}_\ell$. Let Y be the block consisting of blocks Y_1 and Y_2 where t_{Y_2} is a leaf in $\mathcal{T}$ (See Figure 1). Consider the shared vertex x by the blocks X and Y_1. The number of neighbors of x in G' is 2. Thus, by our assumption, x is incident with a vertex in C_n. This contradicts the fact that X be a block in $\mathcal{B}_n \setminus \mathcal{B}_n^{(\geq 3)}$ which is not incident to any edge in C_n. Now to prove the claim the only case remaining is $Y \in \mathcal{B}_n \setminus \mathcal{B}_n^{(\geq 3)}$, but $d_{\mathcal{T}}(t_Y) = 2$ and there is no edge from $C_n^{(\leq 2)}$ incident on Y (See Figure 2). Then, the size of Y is at most 2. Consider the shared vertex x by the blocks X and Y. The number of neighbors of x in G' is 2. Thus by our assumption x is incident with a vertex in C_n. This contradicts the fact that X be a block in $\mathcal{B}_n \setminus \mathcal{B}_n^{(\geq 3)}$ which is not incident to any edge in C_n. This proves the claim. ◀

Using the above claim we can show that the total number of edges in the blocks in $\mathcal{B}_n \setminus \mathcal{B}_n^{(\geq 3)}$ which are not incident to any edge in C_n is bounded by

$$\begin{aligned} 2\left(|\mathcal{C}_n^{(\leq 2)}| + \sum_{\{t \in V(\mathcal{T}) : d_{\mathcal{T}}(t) \geq 3\}} 1\right) &\leq 2|\mathcal{C}_n^{(\leq 2)}| + 2\sum_i r_i \\ &\leq 2|\mathcal{C}_n^{(\leq 2)}| + |C_\ell^{(\leq 4)}| + |C_\ell^{(\geq 5)}| \qquad \text{(By Claim 1)} \qquad (7) \end{aligned}$$

Now, we bound the number of edges in the blocks in $\mathcal{B}_n \setminus \mathcal{B}_n^{(\geq 3)}$ which are incident to some edges in C_n. Let X be a such a block. If the size of X is at least 3, then there are $i-2$ vertices in X such that each of these vertices will have only two neighbors in G'. By our assumption, this implies that there are at least $i-2$ edges from $\mathcal{C}_n^{(\leq 2)}$ which are incident on X. Thus, the total number of edges, in the blocks in $\mathcal{B}_n \setminus \mathcal{B}_n^{(\geq 3)}$, which are not incident to any edge in C_n, is bounded by $3|\mathcal{C}_n^{(\leq 2)}|$. Hence,

$$|B_n \setminus B_n^{(\geq 3)}| = 5|\mathcal{C}_n^{(\leq 2)}| + |C_\ell^{(\leq 4)}| + |C_\ell^{(\geq 5)}| \qquad \text{(By Claim 1)} \qquad (8)$$

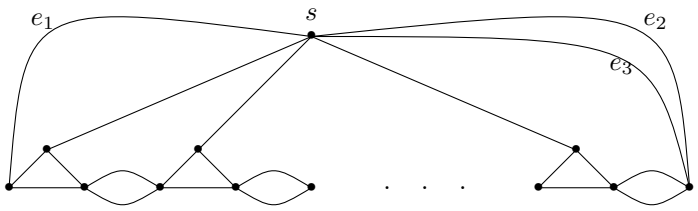


Figure 3 A tight example of Lemma 7. Here $S = \{s\}$.

Hence,

$$\begin{aligned} |B| &= |B_\ell| + |B_n^{(\geq 3)}| + |B_n \setminus B_n^{(\geq 3)}| \\ &= \frac{9}{2}|C_\ell^{(\leq 4)}| + 5|C_\ell^{(\geq 5)}| + 5|\mathcal{C}_n^{(\leq 2)}| + |C_n^{(\geq 3)}| \qquad \text{(By Equations 2,6 and 8)} \\ &\leq 5|C| \end{aligned}$$

This completes the proof of the Lemma. ◀

The bound given in Lemma 7 is in fact tight. Figure 3 represents a family of tight instances. From the figure, let $S = \{s\}$. Let $E_{\mathsf{cross}} = E(S, V(G) \setminus S)$. Let $E' = E_{\mathsf{cross}} - \{e_1, e_2, e_3\}$. Let $E_{\mathsf{cactus}} = E(G - S)$. We see that for every pair of consecutively occuring triangle and double parallel edges in the cactus, there is an edge in E_{cross}. Thus, $|E_{\mathsf{cactus}}| = 5(|E'|)$. This means that $|E_{\mathsf{cactus}}| = 5(|E_{\mathsf{cross}}| - 3)$. Hence, this is a family of tight instances.

4 Algorithm for Even Cycle Transversal

In this section, we give a randomized FPT algorithm for Even Cycle Transversal. This problem is a special case of the following problem.

Parity Even Cycle Transversal	**Parameter:** k
Input: A graph G, a weight function $w : E(G) \to \{0,1\}$ and positive integer k	
Question: Is there a set $S \subseteq V(G)$ of size k such that $G - S$ does not contain any cycle C with $\Sigma_{e \in E(C)} w(e) = 0 \mod 2$?	

We call a cycle C an even-parity (odd-parity) cycle if $\Sigma_{e \in E(C)} w(e) = 0 \mod 2$ ($\Sigma_{e \in E(C} w(e) = 1 \mod 2$). For compactness of notation, we define the function $\mathsf{parity} : 2^{E(G)} \to \{0,1\}$, where for an edge set $E' \in E(G)$, $\mathsf{parity}(E') = \Sigma_{e \in E'} w(e) \mod 2$. In other words, for an even-parity (odd-parity) cycle C, $\mathsf{parity}(E(C)) = 0$ ($\mathsf{parity}(E(C)) = 1$). This should not be confused with cycles of even (odd) length, since we will refer to these cycles simply as even and odd cycles.

In what follows, we give a randomized FPT algorithm for Parity Even Cycle Transversal, that runs in $12^k n^{\mathcal{O}(1)}$ time. First, we preprocess the input graph by applying some reduction rules. A reduction rule reduce an instance (I_1, k) of a problem Π to another instance (I_2, k') of Π. The reduction rule is *safe* when (I_1, k) is a Yes instance if and only if (I_2, k') is a Yes instance. We describe the reduction rules below and prove their safeness. We apply the following rules exhaustively.

▶ **Reduction Rule 1.** *If there is a vertex v in G which is not part of any even-parity cycle, then delete v from G.*

▶ **Lemma 8** (†). *Reduction Rule 1 is safe.*

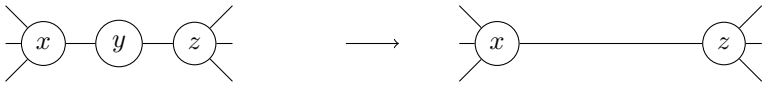


Figure 4 Reduction Rule 2. Here weight of new edge (x, z), $w((x, z)) = (w((x, y)) + w((y, z)) \mod 2$.

In the following Lemma, we show that, on a graph where all edges have weight 1, testing whether a vertex is contained in an even cycle can be done in polynomial time.

▶ **Lemma 9.** *Given a graph G, where every edge has weight 1, and a vertex $v \in V(G)$, there is a polynomial time algorithm that checks whether there is an even cycle containing v.*

Proof. The vertex v is contained in an even cycle C if and only if there is a neighbour $u \in N_G(v)$ such that the edge $(u, v) \in E(C)$. For each $u \in N_G(v)$, we check whether there is an even cycle containing the edge (u, v). This is equivalent to checking whether there is an odd path P between v and w in the graph $G' = G - (u, v)$. In [14], the Parity Multiway Cut (PMWC) problem was posed: If we are given a graph with a set of terminal vertices $T_o \uplus T_e$, does there exist a set S of at most k vertices such that $G - S$ does not have any even path between vertices of T_e and odd paths between vertices of T_o. It was shown that this problem has an FPT algorithm, when parameterised by the size k of the deletion set S. The running time of the algorithm is $2^{2^{\mathcal{O}(k)}} n^{\mathcal{O}(1)}$. We observe that our problem is a special case of the above problem. In our case, $T_o = \{u, v\}, T_e = \emptyset$ and $k = 0$. In other words, we wish to check whether there are any odd paths between u, v in G'. Since $2^{2^{\mathcal{O}(k)}} = \mathcal{O}(1)$, the algorithm for PMWC enables us to check in polynomial time, whether there are no odd paths between u and v in G'. If the algorithm returns Yes, then we know that there are no even cycles in G containing the edge (u, v). Otherwise, we conclude that there is an even cycle in G containing v. If, for every edge $e \in E(G)$ adjacent to v, there is no even cycle containing the edge e, then we conclude that there is no even cycle in G that contains v. ◀

This also gives us a polynomial time algorithm to check whether a vertex of a $(0, 1)$ edge-weighted graph is contained in an even-parity cycle.

▶ **Lemma 10** (†). *Given a graph G, where every edge has weight 0 or 1, and a vertex $v \in V(G)$, there is a polynomial time algorithm that checks whether there is an even-parity cycle containing v.*

▶ **Reduction Rule 2.** *Let $[x, y, z]$ be a path in G and degree of y is exactly 2. Then delete y from G and add a new edge $e_1 = (x, z)$. $w(e_1) = w((x, y)) + w((y, z)) \mod 2$. (See Figure 4).*

▶ **Lemma 11.** *Reduction Rule 2 is safe*

Proof. Suppose C is a cycle of parity p in G, which contains the vertex y. Then, since $d_G(y) = 2$, C must contain the path $[x, y, z]$. In the reduced graph G', C is reduced to a cycle C' which contains the edge $e_1 = (x, z)$. By definition of $w(e_1)$, the parity of the reduced cycle is still p. On the other hand, if C' is a cycle of parity p in the reduced graph G', and C' does not contain the new edge e_1, then C' is a cycle of the original graph G. Otherwise, there is a corresponding cycle C in G, which contains the path $[x, y, z]$ instead of the newly added edge e_1. Again, by definition of $w(e_1)$, the parity of C' and C are the same.

Now, suppose (G, k) is a Yes instance for Parity Even Cycle Transversal. Let S be a solution set in G. Then S hits all even-parity cycles of G. We have argued that any cycle in G that contains y also contains x and z. Thus, if y was contained in S, then

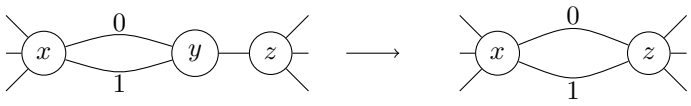


Figure 5 Reduction Rule 3.

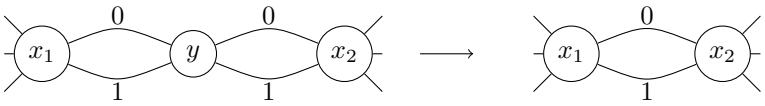


Figure 6 Reduction Rule 4.

$S \cup \{x\} - y$ is also a solution that hits all even-parity cycles of G. Since the parity of cycles is preserved by this reduction, it implies that $S \cup \{x\} - y$ is a solution that hits all even-parity cycles of the reduced graph, and that the reduced instance is also a YES instance.

On the other hand, suppose the reduced instance is a YES instance. let S' be a solution set of G'. We will show that S' is also a solution for G. Suppose there is an even-parity cycle C in G, that is not hit by S', then this cycle must have the vertex y. This implies that the cycle must have the path $[x, y, z]$. Let $P = C - \{y\}$. Look at the cycle $C' = P \cup e_1$ in G'. This is also an even-parity cycle which is not hit by S'. This contradicts the fact that S' is a solution set of G'. Thus, (G, k) must be a YES instance of PARITY EVEN CYCLE TRANSVERSAL. ◀

▶ **Reduction Rule 3.** *Let x, y be two vertices with two parallel edges e_1 and e_2. Let $w(e_1) = 1, w(e_2) = 0$. Further, $e_3 = (y, z)$ is an edge in G, with $z \neq x$, and $d_G(y) = 3$. Then delete y from the graph G and add two new edges $f_1, f_0 = (x, z)$. Define $w(f_1) = 1$ and $w(f_0) = 0$ (See Figure 5).*

▶ **Lemma 12** (†). *Reduction Rule 3 is safe*

▶ **Reduction Rule 4.** *Let $\{x_1, y\}$ be a pair of vertices that have two parallel edges e_1 and e_2, with $w(e_1) = 1, w(e_2) = 0$. Let there be another vertex $x_2 \neq x_1$ such that $\{x_2, y\}$ have two parallel edges e_3 and e_4. It also holds that $w(e_3) = 1, w(e_4) = 0$. Let $d_G(y) = 4$. Then delete y from G and add two new parallel edges f_1, f_0 between x_1 and x_2. We define $w(f_1) = 1$ and $w(f_0) = 0$. (See Figure 6).*

▶ **Lemma 13** (†). *Reduction Rule 4 is safe*

We give the definition of an odd-parity (even-parity) cactus graph and relate it to PARITY EVEN CYCLE TRANSVERSAL.

▶ **Definition 14.** A cactus graph, where the edges have weights from $\{0, 1\}$, is an odd-parity (even-parity) cactus graph when every block of the graph is either an odd-parity (even-parity) cycle or an edge.

▶ **Lemma 15** (†). *Let G be a connected graph and $w : E(G) \to \{0, 1\}$ be a weight function on the edges. G does not contain any cycle C with $w(C) = 0 \mod 2$ if and only if G is an odd-parity cactus.*

Given a graph G, let S be a set of vertices that hits all even-parity cycles. Then each component of $G - S$ does not contain an even-parity cycle. By Lemma 15, it follows that $G - S$ is a forest of odd-parity cacti.

▶ **Observation 1** (†). *Each connected component of the reduced graph for* Parity Even Cycle Transversal *satisfies the conditions of Lemma 7.*

Now, we are ready to describe the algorithm for Parity Even Cycle Transversal.

▶ **Theorem 16.** Parity Even Cycle Transversal *has a randomized algorithm running in* $12^k n^{\mathcal{O}(1)}$ *time.*

Proof. Let S be a solution set of at most k vertices such that $G - S$ is a forest of odd-parity cacti. By Lemma 7, for each component C of G, $|E(C - S)| \leq \frac{5}{6}|E(C \cup S)|$. This implies that $|E(G - S)| \leq \frac{5}{6}|E(G)|$.

Our algorithm is as follows: We define a set $S = \emptyset$ to start with. We pick an edge $e = (u, v) \in E(G)$ uniformly at random and then, with equal probability, we pick one of the two endpoints. We delete this vertex from the current graph and put it into S. In other words, we pick a vertex with probability proportional to its degree. We do this for k steps, at the end of which we check if the constructed set S is a solution set for Parity Even Cycle Transversal. Recognising a forest of odd-parity cacti is equivalent to building a block-decomposition and checking if a block is a odd-parity cycle or an edge. Thus, the entire procedure can be implemented in polynomial time.

Notice that the final set S is a solution set if in each step i, with respect to the current set of vertices in S, we pick a vertex v such that in $G - S$ there is a $k - i$-sized solution set S_i containing v. We will call such a vertex a good vertex for the step i. In step $i \leq k$, the probability, that a good vertex of step i is picked, is at least $\frac{1}{2} \cdot \frac{1}{6} = \frac{1}{12}$. We succeed in finding a solution set S for Parity Even Cycle Transversal if every step picks a good vertex of that step. Thus, the probability of failure in the k-step procedure is at most $1 - (\frac{1}{12})^k$. We repeat the above procedure 12^k times and if in any round we obtain a solution set S of size at most k, we output that set. The probability of failure of this many-round procedure is at most $(1 - (\frac{1}{12})^k)^{12^k} \sim e^{-1}$. The running time of the many-round procedure is $12^k n^{\mathcal{O}(1)}$. ◀

▶ **Corollary 17.** Even Cycle Transversal *has a randomized algorithm running in* $12^k n^{\mathcal{O}(1)}$ *time.*

5 Algorithm for Diamond Hitting Set

In this section, we give a randomized FPT algorithm for Diamond Hitting Set. It was shown in [6] that there is a set of safe reduction rules that can be applied to reduce the input graph to a graph with certain properties.

▶ **Proposition 18** ([6]). *There are polynomial time reduction rules, on application of which, the input instance of* Diamond Hitting Set *is reduced to an equivalent instance where every vertex either has at least three distinct neighbours or three parallel edges.*

▶ **Observation 2** (†). *Each connected component of the reduced graph for* Diamond Hitting Set *satisfies the conditions of Lemma 7.*

Now, we can design an algorithm for Diamond Hitting Set, that is very similar to the algorithm for Parity Even Cycle Transversal.

▶ **Theorem 19** (†). Diamond Hitting Set *has a randomized algorithm running in* $12^k n^{\mathcal{O}(1)}$ *time.*

References

1 Ann Becker, Reuven Bar-Yehuda, and Dan Geiger. Randomized algorithms for the loop cutset problem. *(JAIR)*, 12:219–234, 2000.

2 Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (meta) kernelization. In *FOCS 2009*, pages 629–638, 2009.

3 Yixin Cao. Unit interval editing is fixed-parameter tractable. In *ICALP 2015*, volume 9134 of *LNCS*, pages 306–317. Springer, 2015.

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

5 R. Diestel. *Graph Theory*. Springer, Berlin, second ed., electronic edition, February 2000.

6 Samuel Fiorini, Gwenaël Joret, and Ugo Pietropaoli. Hitting diamonds and growing cacti. In *IPCO 2010*, pages 191–204, 2010.

7 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, Geevarghese Philip, and Saket Saurabh. Hitting forbidden minors: Approximation and Kernelization. In Thomas Schwentick and Christoph Dürr, editors, *STACS 2011*, volume 9 of *(LIPIcs)*, pages 189–200, Dagstuhl, Germany, 2011. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

8 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar f-deletion: Approximation, kernelization and optimal FPT algorithms. In *FOCS 2012*, pages 470–479, 2012.

9 Fedor V. Fomin and Yngve Villanger. Subexponential parameterized algorithm for minimum fill-in. *SIAM J. Comput.*, 42(6):2197–2216, 2013.

10 Archontia C. Giannopoulou, Bart M. P. Jansen, Daniel Lokshtanov, and Saket Saurabh. Uniform kernelization complexity of hitting forbidden minors. In *ICALP 2015*, volume 9134 of *LNCS*, pages 629–641. Springer, 2015.

11 Gwenaël Joret, Christophe Paul, Ignasi Sau, Saket Saurabh, and Stéphan Thomassé. Hitting and harvesting pumpkins. *SIAM J. Discrete Math.*, 28(3):1363–1390, 2014.

12 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. In *ICALP 2013*, volume 7965 of *LNCS*, pages 613–624. Springer, 2013.

13 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is np-complete. *J. Comput. Syst. Sci.*, 20(2):219–230, 1980.

14 Daniel Lokshtanov and M. S. Ramanujan. Parameterized tractability of multiway cut with parity constraints. In *ICALP 2012*, pages 750–761, 2012.

15 Pranabendu Misra, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. Parameterized algorithms for even cycle transversal. In *WG 2012*, pages 172–183, 2012.

16 Carsten Thomassen. On the presence of disjoint subgraphs of a specified type. *Journal of Graph Theory*, 12(1):101–111, 1988.

A Polynomial Kernel for Block Graph Deletion

Eun Jung Kim[1] and O-joung Kwon[2]

1 **LAMSADE, CNRS – Université Paris Dauphine, France**
eunjungkim78@gmail.com

2 **Institute for Computer Science and Control, Hungarian Academy of Sciences, Hungary***
ojoungkwon@gmail.com

Abstract

In the Block Graph Deletion problem, we are given a graph G on n vertices and a positive integer k, and the objective is to check whether it is possible to delete at most k vertices from G to make it a block graph, i.e., a graph in which each block is a clique. In this paper, we obtain a kernel with $\mathcal{O}(k^6)$ vertices for the Block Graph Deletion problem. This is a first step to investigate polynomial kernels for deletion problems into non-trivial classes of graphs of bounded rank-width, but unbounded tree-width. Our result also implies that Chordal Vertex Deletion admits a polynomial-size kernel on diamond-free graphs. For the kernelization and its analysis, we introduce the notion of 'complete degree' of a vertex. We believe that the underlying idea can be potentially applied to other problems. We also prove that the Block Graph Deletion problem can be solved in time $10^k \cdot n^{\mathcal{O}(1)}$.

1998 ACM Subject Classification G.2.1 Combinatorics G.2.2 Graph Theory

Keywords and phrases block graph, polynomial kernel, single-exponential FPT algorithm

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.270

1 Introduction

In parameterized complexity, an instance of a parameterized problem consists in a pair (x, k), where k is a secondary measurement, called the *parameter*. A parameterized problem $Q \subseteq \Sigma^* \times N$ is *fixed-parameter tractable* (*FPT*) if there is an algorithm which decides whether (x, k) belongs to Q in time $f(k) \cdot |x|^{\mathcal{O}(1)}$ for some computable function f. Such an algorithm is called an *FPT* algorithm. We call an FPT algorithm a *single-exponential* FPT algorithm if it runs in time $c^k \cdot |x|^{\mathcal{O}(1)}$ for some constant c. A parameterized problem is said to admit a *polynomial kernel* if there is a polynomial time algorithm in $|x| + k$, called a *kernelization algorithm*, that reduces an input instance into an instance with size bounded by a polynomial function in k, while preserving the Yes/No answer.

Graph modification problems constitute a fundamental class of graph optimization problems. Typically, for a class Φ of graphs, a set Ψ of graph operations and a positive integer k, we want to know whether it is possible to transform an input graph into a graph in Φ by at most k operations chosen in Ψ. One of the most intensively studied graph modification problems is the Feedback Vertex Set problem. Given a graph G and an integer k as input, the Feedback Vertex Set problem asks whether G has a vertex subset of size at most k whose removal makes it a forest, which is a graph without cycles. The Feedback Vertex Set problem is known to admit an FPT algorithm [1, 11] and the

* Supported by ERC Starting Grant PARAMTIGHT (No. 280152).

running time has been subsequently improved by a series of papers [23, 16, 14, 10, 4, 2, 7, 19]. Also, Thomassé [26] showed that it admits a kernel on $O(k^2)$ vertices.

The Feedback Vertex Set problem has been generalized to deletion problems for more general graph classes. *Tree-width* [25] is one of the basic parameters in graph algorithms and plays an important role in structural graph theory. Since forests are exactly the graphs of tree-width at most 1, the natural question is to decide, for an integer $w \geqslant 2$, whether there is an FPT algorithm with parameter k to find a vertex subset of size at most k whose removal makes it a graph of tree-width at most w (called Tree-width w Vertex Deletion). Courcelle's meta theorem [5] implies that the Tree-width w Vertex Deletion is FPT. Recently it is proved to admit a single-exponential FPT algorithm and a (non-uniform) polynomial kernel (a kernel of size $\mathcal{O}(k^{g(w)})$ for some function g) [12, 18].

On the other hand, there are interesting open questions related to two natural graph classes having tree-like structures. A graph is *chordal* if it does not contain any induced cycle of length at least 4. Chordal graphs are close to forests as a forest is a chordal graph without triangles. Marx [20] firstly showed that the Chordal Vertex Deletion problem is FPT, and Cao and Marx [3] improved that it can be solved in time $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(1)}$. However, it remains open whether there is a single-exponential FPT algorithm or a polynomial kernel [20, 3]. Another interesting class is *distance-hereditary graphs*, also known as graphs of *rank-width* at most 1 [22]. As many problems are tractable on graphs of bounded rank-width by the meta-theorem on graphs of bounded rank-width (equivalently, bounded clique-width) [6], it is worth studying the general Rank-width w Vertex Deletion problem. Again, it is known to be FPT from the meta-theorem on graphs of bounded rank-width [6], but for our knowledge, it is open whether there is a single exponential FPT algorithm or a polynomial kernel for this problem even for $w = 1$.

Block graphs lie in the intersection of chordal graphs and distance-hereditary graphs, and they contain all forests. A graph is a *block graph* if each block (maximally 2-connected subgraph) of it forms a clique. It is not difficult to see that block graphs are exactly those not containing an induced cycle of length at least 4 and a diamond (i.e. a cycle of length 4 with a single chord) as an induced subgraph. We study the following parameterized problem.

Block Graph Deletion
Input: A graph G, an integer k
Parameter: k
Question: Is there a vertex subset S of G with $\lvert S\rvert \leqslant k$ such that $G - S$ is a block graph?

Our main results are stated in the next two theorems.

▶ **Theorem 1.1.** *The* Block Graph Deletion *admits a kernel with $O(k^6)$ vertices.*

▶ **Theorem 1.2.** *The* Block Graph Deletion *can be solved in time $10^k \cdot n^{\mathcal{O}(1)}$.*

Our kernelization is motivated by the quadratic vertex-kernel by Thomassé [26]. In [26], basic reduction rules are applied so that whenever the size of the instance is still large, there must be a vertex of large degree (otherwise, it is a No-instance). Then a vertex v of large degree witnesses either so-called the *sunflower* structure, or the *2-expansion* structure. Our kernelization employs a similar strategy. In order to work with block graphs instead of forests, we come up with the notion of the *complete degree* of a vertex, which replaces the role of the usual degree of a vertex in Feedback Vertex Set. Also, we need to bound the size of a block which might appear in a block graph $G - S$, if such a set S of size at most k exists. Our single-exponential algorithm is surprisingly analogous to the algorithm of Chen. *et al.* [4] for Feedback Vertex Set although the analysis is non-trivial.

Since block graphs are exactly diamond-free chordal graphs, we have the following as a corollary of Theorem 1.1 and Theorem 1.2.

▶ **Corollary 1.3.** *On diamond-free graphs,* Chordal Vertex Deletion *admits a kernel with $O(k^6)$ vertices and can be solved in time $O(10^k \cdot n^{\mathcal{O}(1)})$.*

2 Preliminaries

All graphs considered in this paper are undirected and simple (without loops and parallel edges). For a graph G, we denote by $V(G)$ and $E(G)$ the vertex set and the edge set of G, respectively. When we analyze the running time of an algorithm, we agree that $n = |V(G)|$. A *block tree* $\mathcal{T}_G$ of a graph G is the graph having $\mathcal{B} \cup \mathcal{C}$ as the vertex set, where $\mathcal{B}$ is the set of all blocks of G and $\mathcal{C}$ is the set of all cut vertices of G, and there is an edge $Bc \in E(\mathcal{T}_G)$ between $B \in \mathcal{B}$ and $c \in \mathcal{C}$ if and only if the cut vertex c belongs to the block B in G. The constructed graph does not contain a cycle. We say that a graph is a *block graph obstruction*, or simply an *obstruction*, if it is isomorphic to a diamond, or an induced cycle C_ℓ of length ℓ for some $\ell \geqslant 4$. A vertex is *simplicial* in G if $N_G(v)$ is a complete graph.

3 Complete degree of a vertex

We define a concept called the *complete degree* of a vertex in a graph. The definition of the complete degree is motivated by the following lemma, whose proof is deferred at the end of this section.

▶ **Proposition 3.1.** *Let G be a graph and let $v \in V(G)$ and let k be a positive integer. Then in $\mathcal{O}(kn^3)$ time, we can find either ?*
1. *$k+1$ obstructions that are pairwise vertex-disjoint, or*
2. *$k+1$ obstructions whose pairwise intersections are exactly the vertex v, or*
3. *$S_v \subseteq V(G)$ with $|S_v| \leqslant 7k$ such that $G - S_v$ has no block graph obstruction containing v.*

For a graph G and $v \in V(G)$ such that G has no $k+1$ vertex-disjoint obstructions and has no $k+1$ obstructions whose pairwise intersections are exactly the vertex v, the *complete degree* of v is defined as the minimum number of components of $G - (S_v \cup \{v\})$ among all possible $S_v \subseteq V(G) \backslash \{v\}$ where

- $|S_v| \leqslant 7k$, and
- $G - S_v$ has no block graph obstruction containing v.

Note that if $G - S_v$ has no block graph obstruction containing v, then $G[N_G(v) \backslash S_v]$ is a disjoint union of complete graphs.

To prove Proposition 3.1, we use the Gallai's A-path theorem. For a graph G and $A \subseteq V(G)$, an *A-path* of G is a path of length at least 1 whose end vertices are in A, and all internal vertices are in $V(G) \backslash A$.

▶ **Theorem 3.2** (Gallai [13])**.** *Let G be a graph and let $A \subseteq V(G)$ and let k be a positive integer. Then, in $\mathcal{O}(kn^2)$ time, we can find either ?*
1. *$k+1$ vertex-disjoint A-paths, or*
2. *$X \subseteq V(G)$ with $|X| \leqslant 2k$ such that $G - X$ has no A-paths.*

Proof of Proposition 3.1. Let $G_1 := (G - v) - E(G[N_G(v)])$. By Theorem 3.2, we can find in time $\mathcal{O}(kn^2)$ either
1. $2k+1$ vertex-disjoint $N_G(v)$-paths in G_1, or
2. $X \subseteq V(G)$ with $|X| \leqslant 4k$ such that $G_1 - X$ has no $N_G(v)$-paths.

Suppose that G_1 contains at least $2k+1$ pairwise vertex-disjoint $N_G(v)$-paths. Let P be one of these $N_G(v)$-paths in G_1 with p and q as its end vertices, and let P' be a shortest p, q-path

in $G_1[V(P)]$. Note that P' has length at least 2. If P' has length 2, then $G[\{v\} \cup V(P')]$ is isomorphic to either C_4 or the diamond depending on the adjacency between p and q in G. If P' has length at least 3 and $pq \in E(G)$, then $G[V(P')]$ is an induced cycle of length at least 4. If P' has length at least 3 and $pq \notin E(G)$, then $G[\{v\} \cup V(P')]$ is an induced cycle of length at least 5. Thus, $G[\{v\} \cup V(P)]$ contains an obstruction, and G contains either disjoint $k+1$ obstructions, or $k+1$ obstructions whose pairwise intersections are exactly v.

So, we may assume that there exists $X \subseteq V(G_1)$ with $|X| \leqslant 4k$ such that $G_1 - X$ has no $N_G(v)$-paths. Now, we greedily find a maximal set $\mathcal{P}$ of vertex-disjoint induced P_3 in $G[N_G(v)]$ by searching vertex subsets of size 3. If there are $k+1$ vertex-disjoint induced P_3's, then G has $k+1$ diamonds whose pairwise intersections are exactly v. Otherwise, we set $S_v = X \cup \bigcup_{P \in \mathcal{P}} V(P)$ and notice that $|S_v| \leqslant 7k$. Observe that $G - S_v$ has no block graph obstruction containing v. Clearly, we can find $\mathcal{P}$ in time $\mathcal{O}(kn^3)$. ◀

In our algorithm, we need to find a vertex of sufficiently large complete degree and the corresponding deletion set S_v in polynomial time. However, we just need sufficiently many complete graphs on the neighborhood, and do not need to compute the complete degree of each vertex exactly. The following lemma will be used to analyze the difference between an optimal set and an arbitrary set S_v obtained by Proposition 3.1.

▶ **Lemma 3.3.** *Let G be a graph and let $S_1, S_2 \subseteq V(G)$ such that for each $1 \leqslant i \leqslant 2$, $G - S_i$ is a disjoint union of complete graphs. If $|S_2| \leqslant k$, then the number of components of $G - S_2$ is at least the number of components of $G - S_1$ minus k.*

4 Finding a vertex of large complete degree

In this section, we prove that if a graph is reduced under certain rules and its size is still large, then there should exist a vertex of large complete degree. To do this, we first provide basic reduction rules.

4.1 Basic reduction rules

▶ **Reduction Rule 1** (Block component rule). *If G has a component H that is a block graph, then we remove H from G.*

▶ **Reduction Rule 2** (Cut vertex rule). *Let v be a vertex of G such that $G - v$ contains a component H where $G[V(H) \cup \{v\}]$ is a connected block graph. Then we remove H from G.*

Two vertices v, w in a graph G are called *true twins* if $N_G(v) \backslash \{w\} = N_G(w) \backslash \{v\}$ and $vw \in E(G)$. Note that two simplicial vertices in a block of a block graph are true twins.

▶ **Reduction Rule 3** (Twin rule). *Let S be the set of vertices that are pairwise true twins in G. If $|S| \geqslant k+2$, then we remove vertices except $k+1$ vertices.*

It is not hard to observe that Rules 1, 2, and 3 are sound. Note that we can test whether a given graph is a block graph in quadratic time using an algorithm to partition the graph into blocks [15], and testing whether each block is a complete graph.

▶ **Reduction Rule 4** (Reducing block-cut vertex paths). *Let $t_1t_2t_3t_4$ be an induced path of G and for each $1 \leqslant i \leqslant 3$, let $S_i \subseteq V(G) \backslash \{t_1, \ldots, t_4\}$ be a clique of G such that*

- *for each $1 \leqslant i \leqslant 3$ and $v \in S_i$, $N_G(v) \backslash S_i = \{t_i, t_{i+1}\}$, and*
- *for each $2 \leqslant i \leqslant 3$, $N_G(t_i) = \{t_{i-1}, t_{i+1}\} \cup S_{i-1} \cup S_i$.*

Then we remove S_2 and contract t_2t_3.

Clearly, we can apply Reduction Rule 4 in polynomial time. We prove the soundness of Reduction Rule 4 in the full version [17]. The following rule will be applied using Proposition 3.1.

▶ **Reduction Rule 5** ($(k+1)$-distinct obstructions rule). *Let $v \in V(G)$ and let $G' := G - v - E(G[N_G(v)])$ such that there are $2k+1$ vertex-disjoint $N_G(v)$-paths in G'. If G contains $k+1$ vertex-disjoint obstructions, then say that it is a* No*-instance. Otherwise, we remove v from G, and decrease k by one. (By Proposition 3.1, one of them exists.)*

4.2 A vertex of large complete degree

An instance (G, k) is called a *reduced instance* if it is reduced under Rules 1, 2, 3, 4, and 5 introduced in the previous subsection. In this subsection, we prove that there exists a vertex of large complete degree whenever a reduced instance is sufficiently large, which is stated as Theorem 4.1.

For positive integers k, ℓ, we define that

- $g_1(k,\ell) := 6k^2(\ell+14k)^2 + 2k(\ell+14k)$,
- $g_2(k,\ell) := (k+1)^2 + 7k^2 + \frac{1}{2}k(\ell+14k)$.

▶ **Theorem 4.1.** *Let (G,k) be a reduced instance of* Block Graph Deletion *that is a* Yes*-instance. If G has at least $k + g_1(k,\ell)g_2(k,\ell)$ vertices then G has a vertex of complete degree at least $\ell+1$.*

Let (G,k) be a reduced instance of Block Graph Deletion and let $S \subseteq V(G)$ of size at most k such that $G-S$ is a block graph. We let $G' := G - S$ and for each $v \in S$, we define that

- $G_v := G[V(G') \cup \{v\}]$,
- S'_v is a vertex set of size at most $7k$ in $G-v$ that is obtained by Proposition 3.1,
- $S_v := S'_v \cap V(G')$.

Let $T := \bigcup_{v \in S} S_v$. Note that $|T| \leqslant 7k^2$ and for each $v \in S$, there are no block graph obstructions containing v in $G_v - T$.

We first give a bound on the size of each block of G' and the number of blocks in G' sharing a cut vertex with it, assuming that there is no vertex in S of large complete degree in G. Each block of G' consists of the set of simplicial vertices and the set of cut vertices in G'.

▶ **Lemma 4.2.** *Let F be a graph whose vertex set is $X \cup \{v_1, \ldots, v_t\}$ such that $t \geqslant 2$ and X is a clique of F and every two vertices of X have different neighbors on $\{v_1, \ldots, v_t\}$. If $|X| \geqslant t+2$, then F contains a diamond having exactly one vertex of $\{v_1, \ldots, v_t\}$.*

Proof. Without loss of generality, we can assume that $\{v_1, \ldots, v_t\}$ is a minimal set with the aforementioned property. Notice that there exists a vertex v_i which has at least two neighbors in X. By minimality assumption, v_i is not adjacent with all vertices in X. Choose distinct vertices $x, y, z \in X$ such that x,y are neighbors of v_i and z is not. Observe that $F[\{v_i, x, y, z\}]$ is isomorphic to the diamond containing exactly one vertex of $\{v_1, \ldots, v_t\}$ ◀

▶ **Lemma 4.3.** *Let B be a block of G', and let B_1 and B_2 be the sets of all simplicial vertices and all cut vertices of G' contained in B, respectively. Let $H_1, H_2, \ldots, H_t$ be the components of $G' - V(B)$ that has a neighbor in B. The followings hold.* ?

1. $|B_1| \leqslant (k+1)^2 + 7k^2$.
2. *If for every $v \in S$, v has complete degree at most ℓ in G, then $|B_2| \leqslant t \leqslant k(\ell+14k)$.*

Proof. The proof for (2) can be found in the full version [17]. (1) We first give a bound on the number of simplicial vertices of a block for $G' - T$. Note that $B \backslash T$ is a block of $G' - T$. Let B'_1 be $B_1 \backslash T$. Clearly, $|B_1| \leqslant |B'_1| + 7k^2$.

Since vertices in B'_1 are pairwise true twins in G', if two vertices in B'_1 have the same neighbors on S, then they are true twins in G. We partition B'_1 into an equivalent classes where two vertices are equivalent if they have the same neighbors on S. From Reduction Rule 3, each equivalent class has at most $k + 1$ vertices.

If $|S| \leqslant 1$, then there are at most 2 equivalent classes in B'_1. If $|S| \geqslant 2$ and the number of equivalent classes in B'_1 is at least $k + 2$, then since $|S| \leqslant k$, G contains a diamond containing exactly one vertex v of S by Lemma 4.2. This contradicts to the fact that $G_v - T$ has no obstruction containing v. Thus, the number of equivalent classes in B'_1 is at most $k + 1$ and $|B_1| \leqslant |B'_1| + 7k^2 \leqslant (k + 1)^2 + 7k^2$. ◀

Contracted Block Tree. We introduce a notion called the *contracted block tree* of G. A contracted block tree $\mathcal{T}_G$ of a connected graph G is a rooted tree obtained from a block tree $\mathcal{T}^0_G$ of G by (i) choosing a block vertex of $\mathcal{T}^0_G$ as a root, and (ii) for each cut vertex c of $\mathcal{T}^0_G$, identifying it with its unique parent.

Let $\mathcal{T}_{G'}$ be the union of contracted block trees of connected components of G'. We color the vertices of $\mathcal{T}_{G'}$ in three phases: in the first phase, for every vertex $v \in S$ and for every $w \in N_{V(G) \setminus S}(v)$, we choose the (unique) block $B \in V(\mathcal{T}_{G'})$ which contains w and is closest to the root, and color B by red. Let R_1 be the vertices colored red so far. In the second phase, we again recursively color the least common ancestor of any pair of red vertices by red. Let R be the set of red vertices $\mathcal{T}_{G'}$. All other vertices of $\mathcal{T}_{G'}$ are colored blue.

▶ **Lemma 4.4.** *Suppose that the complete degree of v is at most ℓ for every $v \in S$. Then we have $|R| \leqslant 2k(\ell + 14k)$.*

▶ **Lemma 4.5.** *Let T be a tree with at least 2 vertices and degree at most d, and let M be a set of vertices in T. Then there are at most $d \cdot |M|$ connected components in $T - M$.*

The next lemma follows from Lemma 4.4 and 4.5.

▶ **Lemma 4.6.** *If G' contains at least $g_1(k, \ell)$ blocks, then $\mathcal{T}_{G'}$ has a blue component on at least 3 vertices.*

Proof of Theorem 4.1. Let (G, k) be a reduced instance with $|V(G)| \geqslant k + g_1(k, \ell)g_2(k, \ell)$ and $S \subseteq V(G)$ be a set of size at most k such that $G - S$ is a block graph. To derive contradiction, suppose that for every $v \in S$, v has complete degree at most ℓ in G. Then $G' = G - S$ has at least $g_1(k, \ell)g_2(k, \ell)$ vertices. Let p be the number of blocks of G'. From Lemma 4.3 and the fact that each cut vertex is contained in at least two blocks, we obtain $|V(G')| \leqslant p((k + 1)^2 + 7k^2) + \frac{1}{2}pk(\ell + 14k) \leqslant p \cdot g_2(k, \ell)$. Therefore, we have $p \geqslant g_1(k, \ell)$. By Lemma 4.6, $\mathcal{T}_{G'}$ contains a blue component P on at least 3 vertices.

We claim that P is (i) a path, and (ii) each of its two end vertices, and no other, is adjacent with exactly one red vertex. Let us prove (i) first. Let W be the unique block vertex in P which is closest to the root. Notice that W is not the root itself since the instance is reduced with respect to Reduction Rule 1 and thus the root is a red vertex. Hence W has a unique parent which is red. For any Z which is a leaf in the subtree P, it is adjacent with at least one red vertex. Indeed, if not, Z is a leaf in $\mathcal{T}_{G'}$. Then by Reduction Rule 2, the block Z (possibly except for its unique cut vertex) should have been removed from G, a contradiction. Note that any red vertex adjacent with Z is a child of Z since the path from Z to W is blue and $W \neq Z$. Furthermore, the subtree P has exactly one leaf since

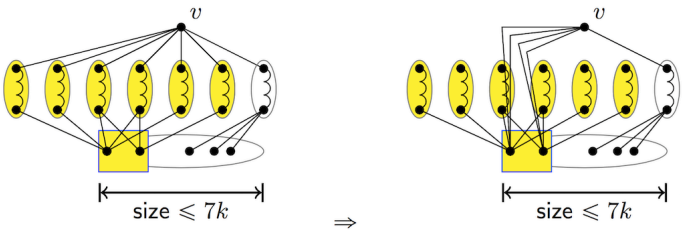


Figure 1 Reduction Rule 6.

otherwise, the second phase of coloring must have colored the branching vertices contained in P, a contradiction. This establishes (i). For (ii), observe that if (ii) does not hold, then some vertex of P must have been colored in the second phase, a contradiction.

Now, with P together with the two red vertices incident with $V(P)$, we can apply Reduction Rule 4, a contradiction. Therefore, we conclude that there exists a vertex $v \in S$ such that v has complete degree at least $\ell + 1$ in G. ◀

5 Reducing the instance with large complete degree

We introduce the last rule, which will be used when G has a vertex of large complete degree. We use the well-known technique, called the α-expansion lemma, which is already used in several kernelization algorithms [26, 9, 21, 8]. One notable difference from other approaches is that, to guarantee the equivalence, we add some paths in the given graph, and thus increase the number of vertices. However, we show that our rule decreases $n + m$ where m is the number of edges whose both degrees are at least 3, by using the 3-expansion le???mma instead of the 2-expansion lemma.

▶ **Reduction Rule 6** (Large complete degree rule)**.** *Let $v \in V(G)$ and $X \subseteq V(G)\backslash\{v\}$ with $|X| \leqslant 7k$. Let $\mathcal{C}$ be a set of connected components of $G - (X \cup \{v\})$ and let $\phi : X \to \binom{\mathcal{C}}{3}$ such that*

- *for each $C \in \mathcal{C}$, $G[\{v\} \cup V(C)]$ is a block graph, v has a neighbor in C, and there exists a vertex $x \in X$ that has a neighbor in C,*
- *for $x \in X$, $\phi(x)$ is a subset of $\mathcal{C}$ where each graph in $\phi(x)$ has a neighbor of x, and*
- *the sets in $\{\phi(x) : x \in X\}$ are pairwise disjoint.*

Then, remove all edges between v and every component of $\mathcal{C}$, and add two internally vertex-disjoint paths of length two between v and each vertex $x \in X$. (All of the new vertices in these paths have degree 2 in the resulting graph). If a component of $\mathcal{C}$ has a vertex of degree 1 in the resulting graph, then we remove the vertex. See Figure 1.

We prove that Reduction Rule 6 is safe in the full version [17]. As we discussed, we clarify that it decreases $n + m^*$ where m^* is the number of edges whose both end vertices have degree at least 3. Since $|\mathcal{C}| \geqslant 3|X|$ and $n + m^*$ is increased by $2|X|$ by adding paths of length 2 from v to each vertex of X, it is sufficient to show that for each $C \in \mathcal{C}$, $n + m^*$ is decreased by at least 1 by removing the edges between v and C. Let $C \in \mathcal{C}$. If $|N_G(v) \cap C| \geqslant 3$, then it is trivial. First assume that $|N_G(v) \cap C| = 2$. Then C has more than two vertices, or there exists a vertex $x \in X$ that has a neighbor on $N_G(v) \cap C$. In either case, it is not difficult to verify that one of the vertex in $N_G(v) \cap C$ has degree at least 3 in G. Therefore, m^* is decreased by at least 1 when removing the edges between v and C. Now, let us assume that $N_G(v) \cap C = \{w\}$ for some $w \in V(C)$. If w has degree 2, then after removing the edge

vw, we also remove w following Reduction Rule 6. Thus, n is decreased by 1. Otherwise, removing vw decreases m^* by 1. We conclude that $n+m^*$ is always decreased when applying Reduction Rule 6.

Now we describe how to obtain a polynomial-size kernel from a given instance. The algorithm presented in the following theorem is used as a subroutine.

▶ **Theorem 5.1** (α-expansion lemma [26]). *Let α be a positive integer. Let F be a bipartite graph on the bipartition (X,Y) with $|Y| \geqslant \alpha|X|$ such that every vertex of Y has at least one neighbor in X. Then there exist nonempty subsets $X' \subseteq X$ and $Y' \subseteq Y$ and a function $\phi : X' \to \binom{Y'}{\alpha}$ such that*

- $N_F(Y') \cap X = X'$,
- $\phi(x) \subseteq N_F(x)$ *for each* $x \in X'$, *and*
- *the sets in* $\{\phi(x) : x \in X'\}$ *are pairwise disjoint.*

In addition, such pair of subsets X', Y' can be computed in polynomial time in $\alpha|V(F)|$.

Proof of Theorem 1.1. Given an instance (G,k), we exhaustively apply Reduction Rules 1-5 to obtain a reduced instance. If a reduced graph G has at least $k+g_1(k,29k)g_2(k,29k)$ vertices, then by Theorem 4.1, G has a vertex of complete degree at least $29k$. By Proposition 3.1, we can find in polynomial time a vertex v and a vertex set $S_v \subseteq V(G-v)$ such that $G-S_v$ has no block graph obstruction containing v, and $G[N_G(v)\backslash S_v]$ has at least $29k-7k=22k$ components. Note that there are at most k components of $G-(\{v\}\cup S_v)$ that may contain an obstruction, and for each component C of $G-(\{v\}\cup S_v)$, at most one components of $G[N_G(v)\backslash S_v]$ can be contained in C. Let $\mathcal{C}$ be the set of components of $G-(\{v\}\cup S_v)$ which (i) contains a component of $G[N_G(v)\backslash S_v]$, and (ii) has no block graph obstructions. Since $|\mathcal{C}| \geqslant 22k-k=21k$ and $|S_v| \leqslant 7k$, using Theorem 5.1, we can find in polynomial time sets $\mathcal{C}' \subseteq \mathcal{C}$ and $S'_v \subseteq S_v$ and a function $\phi : S'_v \to \binom{\mathcal{C}'}{3}$ such that

- the set of vertices in S_v that has a neighbor in $\bigcup_{C\in\mathcal{C}'} V(C)$ is S'_v,
- for $x \in S'_v$, $\phi(x)$ is a subset of $\mathcal{C}$ where each graph in $\phi(x)$ has a neighbor of x, and
- the sets in $\{\bigcup_{C\in\phi(x)} V(C) : x \in S'_v\}$ are pairwise disjoint.

Note that for each $C \in \mathcal{C}'$, $G[\{v\}\cup V(C)]$ is a block graph, otherwise, it has an obstruction containing v, contradicting to the definition of S_v. Furthermore, for each $C \in \mathcal{C}'$, there exists a vertex $x \in S'_v$ that has a neighbor in C, otherwise, we can reduce it using Reduction Rule 2. So, we can apply Reduction Rule 6 to reduce this instance. We apply these reductions recursively. As we discussed, each step decreases $n+m^*$ where m^* is the number of edges whose both end vertices have degree 3, so, it will terminate in polynomial time, and at the final step, the resulting graph will have less than $k+g_1(k,29k)g_2(k,29k) = \mathcal{O}(k^6)$ vertices. ◀

6 A fixed parameter tractable algorithm

The goal of this section is to prove Theorem 1.2 claiming an $O(10^k \cdot n^{O(1)})$-time algorithm for Block Graph Deletion. We apply iterative compression technique, which is established as a powerful tool to design FPT algorithms since it was first introduced by Reed, Smith and Vetta [24]. Our algorithm Block Graph Deletion requires as a subroutine an FPT algorithm for the following disjoint version of Block Graph Deletion.

DISJOINT BLOCK GRAPH DELETION
Input: A graph G, $S \subseteq V(G)$ such that both $G - S$ and $G[S]$ are block graphs, an integer k.
Parameter: k
Task: Find a *solution* to (G, S, k), i.e. a set $\tilde{S} \subseteq V(G) \backslash S$ such that $G - \tilde{S}$ is a block graph and $|\tilde{S}| \leqslant k$, or correctly report that no such set exists.

We present an algorithm **Block**(G, S, k) which solves DISJOINT BLOCK GRAPH DELETION in time $O(3^{k+\ell} \cdot n^6)$, where ℓ is the number of connected components in $G[S]$.

Algorithm 1 Algorithm for BLOCK GRAPH DELETION

```
1:  procedure Block(G, S, k)
2:      if k ⩽ 0 and V(G)\S ≠ ∅, return No.
3:      if k ⩾ 0 and G is a block graph, return ∅.
4:      if u, v, w ∈ V(G)\S s.t. G[S ∪ {u, v, w}] is not a block graph then
5:                                  ▷ u, v, w are not necessarily distinct if |V(G)\S| ⩽ 2
6:          Block(G − u, S, k − 1) ∪ {u}                  ▷ Small Set Branching Rule
7:          Block(G − v, S, k − 1) ∪ {v}
8:          Block(G − w, S, k − 1) ∪ {w}
9:      else if there is uv ∈ E(G − S) and x, y ∈ N_S({u, v}) s.t.
10:         x, y belong to distinct connected components of G[S] then
11:         Block(G − u, S, k − 1) ∪ {u}                  ▷ Component Branching Rule
12:         Block(G − v, S, k − 1) ∪ {v}
13:         Block(G, S ∪ {u, v}, k)
14:     else
15:         Let B be a leaf block of G − S and ∂_{G−S}(B) = {b}.
16:         G' ← G − B\∂_{G−S}(B) + {bw : w ∈ N_S(B)}      ▷ Bypass Rule
17:         Block(G', S, k).
18:     end if
19: end procedure
```

Let us establish that **Block**(G, S, k) correctly returns a solution to (G, S, k) if it is a YES-instance, and returns NO otherwise. Notice that if (G, S, k) does not meet the condition at line 3, then $V(G)\backslash S$ is non-empty and thus one of the steps at lines 2, 4, 10, or 15 will be executed and some output will be returned at the end of the algorithm **Block**(G, S, k). The execution of **Block**(G, S, k) can be represented by a search tree where each node corresponds to a call made during the execution. For the correctness of the algorithm, we use induction on the level of a call in the search tree. It is clear that lines 2–3, corresponding to the base case, returns the output correctly. If the condition at line 4 is met, then any solution $\tilde{S}$ to (G, S, k) must contain one of u, v and w. Conversely, if $\tilde{S}$ is a solution returned by one of the calls **Block** at lines 6–8, then $\tilde{S}$ together with u, v, or w is a solution to (G, S, k). To see the correctness of lines 11–13, first notice that they enumerate all possible intersection of a solution $\tilde{S} \cap \{u, v\}$. Hence it suffices to verify that $G[S \cup \{u, v\}]$ is indeed a block graph. This is a consequence from the fact that G does not meet the condition of line 4 for any (at most) three vertices.

The branching rules considered at lines 4-8 and lines 10-13 are called the `Small Set Branching` and `Component Branching`, respectively. Notice that an instance (G, S, k) considered at line 15 is *reduced with respect* to `Small Set Branching` and `Component Branching` or, simply put, *irreducible*: neither branching rules apply to (G, S, k). For the correctness of

the algorithm **Block**, it remains to show that `Bypass Rule` at line 16 is safe, that is, $\tilde{S}$ is a solution to the instance (G', S, k) at line 17 if and only if it is a solution to (G, S, k). We need the following lemmata, whose proofs can be found in the full version [17].

▶ **Lemma 6.1.** *Let (G, S, k) be an irreducible instance and B be a leaf block of $G - S$. Then either $N_S(B) = \varnothing$ or there exists a single block X of $G[S]$ such that $N_S(B) \subseteq X$.*

▶ **Lemma 6.2.** *Let (G, S, k) be an irreducible instance and B be a leaf block of $G - S$. Then $G[S \cup B]$ is a block graph.*

▶ **Lemma 6.3.** *Let (G, S, k) be an irreducible instance and B be a leaf block of $G - S$. Then there exists a vertex $u \in B$ such that $N_S(u) = N_S(B)$.*

▶ **Lemma 6.4.** *Let (G, S, k) be an irreducible instance and B be a leaf block of $G - S$. If there is a vertex set $\tilde{S} \subseteq V(G) \setminus S$ such that $G - \tilde{S}$ is a block graph, then there is $\tilde{S}' \subseteq V(G) \setminus S$ such that $G - \tilde{S}'$ is a block graph, $|\tilde{S}'| \leqslant |\tilde{S}|$ and $\tilde{S}' \cap (B \setminus \partial_{G-S}(B)) = \varnothing$.*

The following lemma states the correctness of `Bypass Rule` applied at lines 15–17.

▶ **Lemma 6.5.** *Let (G, S, k) be an irreducible instance, B be a leaf block of $G - S$, and G' be the graph obtained by applying `Bypass Rule`. Then,*

- *if $\tilde{S}$ is a solution to (G, S, k), $\tilde{S} \setminus (B \setminus \partial_{G-S}(B))$ is a solution to (G', S, k), and*
- *if $\tilde{S}'$ is a solution to (G', S, k), it is also a solution to (G, S, k).*

Proof. Let b be the unique cut vertex of $G - S$ contained in B. Let us prove the first implication. Suppose that $\tilde{S}$ is a solution to (G, S, k) such that $\tilde{S} \cap (B \setminus \partial_{G-S}(B)) = \varnothing$. Such a solution exists by Lemma 6.4. We show that $\tilde{S}$ is a solution to (G', S, k), from which the first implication follows. If $b \in \tilde{S}$, then $G' - \tilde{S}$ clearly a block graph as it is an induced subgraph of $G - \tilde{S}$. Let us consider the case when $b \notin \tilde{S}$. For the sake of contradiction, suppose that $G' - \tilde{S}$ contains a vertex set C inducing an obstruction. Consider a vertex $u \in B$ such that $N_S(u) = N_S(B)$. The existence of such u is shown in Lemma 6.3. Note that $u \neq b$ and there exists $x \in N_S(B)$ such that $bx \notin E(G)$ and bx is contained in C, otherwise, C also appears in $G - \tilde{S}$. If C contains one more vertex from $N_S(B)$, then C should be a diamond with two intersections on $N_S(B)$ in $G' - \tilde{S}$. Then $G[V(C) \setminus \{b\} \cup \{u\}]$ is a diamond of $G - \tilde{S}$, which is a contradiction. Thus, $|V(C) \cap N_S(B)| = 1$ and $G[V(C) \cup \{u\}]$ induces a subgraph isomorphic to a graph obtained from C by subdividing one edge. It contains an obstruction in $G - \tilde{S}$, which contradicts to our assumption.

We establish the second implication. Suppose that $\tilde{S}'$ is a solution to (G', S, k), but $G - \tilde{S}'$ is not a block graph. Let C be a vertex set inducing an obstruction in $G - \tilde{S}'$. Then $G[C]$ is not a diamond nor a cycle of length 4 since otherwise, $G[C \cup S]$ is not a block graph and $|C \setminus S| \leqslant 3$, contradicting to the assumption that (G, S, k) is reduced with respect to `Small Set Branching`. Therefore $G[C]$ must be an induced cycle of length at least 5. Notice that C contains some vertex $v \notin B \cup S$ since $G[B \cup S]$ is a block graph by Lemma 6.2. There are two possibilities, and in each case we derive a contradiction.

When $b \notin C$: Notice that $N_S(B) \cap C$ is a separator between $B \cap C$ and v in $G[C]$, and thus contains a minimal separator between $B \cap C$ and v. However, $N_S(B)$ is a complete graph by Lemma 6.1 while any minimal separator in an induced cycle must be non-adjacent, a contradiction.

When $b \in C$: Observe that there is a vertex $x \in N_S(B) \cap C$ such that x is adjacent with some vertex, say w, in $B \cap C$. We claim that $N_S(B) \cap C = \{x\}$. Suppose not, and let y be a vertex in $(N_S(B) \cap C) \setminus \{x\}$. The existence of $v \in C \setminus (B \cup S)$ implies $wy \notin E(G)$. Take $u \in B$ such that $N_S(u) = N_S(B)$, which is possible due to Lemma 6.3, and observe that $ux, uy \in E(G)$. It follows that $G[\{u, w, x, y\}]$ is a diamond, contradicting to the assumption that (G, S, k) is reduced with respect to `Small Set Branching`. From $\{x\} \subseteq N_S(B) \cap C$, our claim follows. Notice that $|C \cap B| \leqslant 2$ since an induced cycle can intersect with a clique in at most two vertices. Therefore, $(C \setminus B) \cup \{b\}$ has at least four vertices. Also $G'[(C \setminus B) \cup \{b\}]$ is an induced cycle as no chord can be added in the construction of G' from G. This contradicts to the assumption that $G' - \tilde{S}'$ is a block graph. This completes the proof of the lemma. ◀

▶ **Lemma 6.6.** *Given an instance (G, S, k) to* Disjoint Block Graph Deletion *with $n = |V(G)|$, the algorithm* **Block**(G, S, k) *correctly returns a solution or outputs* No *in time $O(3^{k+\ell} \cdot n^6)$.*

Proof. The correctness of the algorithm is discussed above. For the analysis of the running time, we use the fact that every branching during the execution of **Block**(G, S, k) decreases either k or the number of connected components in $G[S]$ by at least one. The details of the proof can be found in the full version [17]. ◀

Finally, to solve Block Graph Deletion, we apply the standard iterative compression technique. Together with the algorithm **Block** for Disjoint Block Graph Deletion and its analysis given in Lemma 6.6, we obtain an FPT algorithm stated in Theorem 1.2. The full proof is given in the full version [17].

References

1 Hans L. Bodlaender. On disjoint cycles. In *Proceedings of the 17th International Workshop*, WG'91, pages 230–238, London, UK, UK, 1992. Springer-Verlag.

2 Yixin Cao, Jianer Chen, and Yang Liu. On feedback vertex set: New measure and new structures. *Algorithmica*, pages 1–24, 2014.

3 Yixin Cao and Dániel Marx. Chordal editing is fixed-parameter tractable. In *31st International Symposium on Theoretical Aspects of Computer Science*, volume 25 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages 214–225. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2014.

4 Jianer Chen, Fedor V. Fomin, Yang Liu, Songjian Lu, and Yngve Villanger. Improved algorithms for feedback vertex set problems. *J. Comput. System Sci.*, 74(7):1188–1198, 2008.

5 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inform. and Comput.*, 85(1):12–75, 1990.

6 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.

7 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time (extended abstract). In *2011 IEEE 52nd Annual Symp. on Foundations of Computer Science (FOCS'11)*, pages 150–159. IEEE CS, 2011.

8 Marek Cygan, Marcin Pilipczuk, Michał Pilipczuk, and Jakub Onufry Wojtaszczyk. Subset feedback vertex set is fixed-parameter tractable. *SIAM Journal on Discrete Mathematics*, 27(1):290–309, 2013.

9 Marek Cygan, Marcin Pilipczuk, Michał Pilipczuk, and JakubOnufry Wojtaszczyk. An improved FPT algorithm and quadratic kernel for pathwidth one vertex deletion. In Venkatesh Raman and Saket Saurabh, editors, *Parameterized and Exact Computation*, volume 6478 of *Lecture Notes in Computer Science*, pages 95–106. Springer Berlin Heidelberg, 2010.

10 Frank Dehne, Michael Fellows, Michael Langston, Frances Rosamond, and Kim Stevens. An $O(2^{O(k)}n^3)$ fpt algorithm for the undirected feedback vertex set problem. *Theory of Computing Systems*, 41(3):479–492, 2007.

11 Rod Downey and Michael Fellows. Fixed-parameter tractability and completeness. III. Some structural aspects of the W hierarchy. In *Complexity theory*, pages 191–225. Cambridge Univ. Press, Cambridge, 1993.

12 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 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 470–479, 2012.

13 T. Gallai. Maximum-minimum Sätze und verallgemeinerte Faktoren von Graphen. *Acta Math. Acad. Sci. Hungar.*, 12:131–173, 1961.

14 Jiong Guo, Jens Gramm, Falk Hüffner, Rolf Niedermeier, and Sebastian Wernicke. Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. *Journal of Computer and System Sciences*, 72(8):1386–1396, 2006.

15 John Hopcroft and Robert Tarjan. Algorithm 447: Efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378, June 1973.

16 Iyad Kanj, Michael Pelsmajer, and Marcus Schaefer. Parameterized algorithms for feedback vertex set. In *Parameterized and Exact Computation*, volume 3162 of *LNCS*, pages 235–247. Springer, 2004.

17 Eun Jung Kim and O-joung Kwon. A polynomial kernel for Block Graph Vertex Deletion, 2015. arXiv.org:abs:1506.08477.

18 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. In *Automata, Languages, and Programming – 40th Int'l Colloquium, ICALP 2013, Proceedings, Part I*, pages 613–624, 2013.

19 Tomasz Kociumaka and Marcin Pilipczuk. Faster deterministic feedback vertex set. *Information Processing Letters*, 114(10):556–560, 2014.

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

21 Neeldhara Misra, Geevarghese Philip, Venkatesh Raman, and Saket Saurabh. On parameterized independent feedback vertex set. *Theoretical Computer Science*, 461(0):65–75, 2012. 17th International Computing and Combinatorics Conference (COCOON 2011).

22 Sang-il Oum. Rank-width and vertex-minors. *J. Combin. Theory Ser. B*, 95(1):79–100, 2005.

23 Venkatesh Raman, Saket Saurabh, and C. R. Subramanian. Faster fixed parameter tractable algorithms for undirected feedback vertex set. In *Algorithms and computation*, volume 2518 of *Lecture Notes in Comput. Sci.*, pages 241–248. Springer, Berlin, 2002.

24 Bruce Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32(4):299–301, 2004.

25 Neil Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.

26 Stéphan Thomassé. A quadratic kernel for feedback vertex set. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 115–119. ACM/SIAM, 2009.

Parameterized Complexity of Graph Constraint Logic

Tom C. van der Zanden

Department of Computer Science, Utrecht University, Utrecht, The Netherlands
T.C.vanderZanden@uu.nl

Abstract

Graph constraint logic is a framework introduced by Hearn and Demaine [6], which provides several problems that are often a convenient starting point for reductions. We study the parameterized complexity of Constraint Graph Satisfiability and both bounded and unbounded versions of Nondeterministic Constraint Logic (NCL) with respect to solution length, treewidth and maximum degree of the underlying constraint graph as parameters. As a main result we show that restricted NCL remains *PSPACE*-complete on graphs of bounded bandwidth, strengthening Hearn and Demaine's framework. This allows us to improve upon existing results obtained by reduction from NCL. We show that reconfiguration versions of several classical graph problems (including independent set, feedback vertex set and dominating set) are *PSPACE*-complete on planar graphs of bounded bandwidth and that Rush Hour, generalized to $k \times n$ boards, is *PSPACE*-complete even when k is at most a constant.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Nondeterministic Constraint Logic, Reconfiguration Problems, Parameterized Complexity, Treewidth, Bandwidth

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.282

1 Introduction

Nondeterministic Constraint Logic (NCL) was introduced by Hearn and Demaine in [6] and extended in [8] to a more general graph constraint logic framework. The framework provides a number of problems complete for various complexity classes, that aim to provide a convenient starting point for reductions proving the hardness of games and puzzles. We study the Constraint Graph Satisfiability problem and Nondeterministic Constraint Logic in a parameterized setting, considering (combinations of) solution length, treewidth and maximum degree of the underlying constraint graph as parameters.

As part of the constraint logic framework [6], Hearn and Demaine provide a restricted variant of Nondeterministic Constraint Logic (restricted NCL), in which the constraint graph is planar, 3-regular, uses only weights in $\{1, 2\}$ and the graph is constructed from only two specific vertex types (AND and OR). Restricted NCL is *PSPACE*-complete, and is (due to the restrictions) a particularly suitable starting point for reductions. Hearn and Demaine's reduction creates graphs of unbounded treewidth. We strengthen their result, by providing a new reduction showing that restricted NCL remains *PSPACE*-complete, even when restricted to graphs of bandwidth at most a given constant (which is a subclass of graphs of treewidth at most a given constant). We show hardness by reduction from H-Word Reconfiguration [14].

The puzzle game Rush Hour, when generalized to $n \times n$ boards, is *PSPACE*-complete [4]. Hearn and Demaine provide a reduction from NCL to Rush Hour [7]. As a consequence of this reduction and our improved hardness result for NCL, we show that Rush Hour is

10th International Symposium on Parameterized and Exact Computation (IPEC 2015).
Editors: Thore Husfeldt and Iyad Kanj; pp. 282–293

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

Table 1 Parameterized complexity of graph constraint logic problems.

		Problem				
		CGS	C2E	C2C	BC2E	BC2C
Parameters	–	*NP*-C	*PSPACE*-C	*PSPACE*-C	*NP*-C	*NP*-C
	l	–	*W*[1]-hard	*W*[1]-hard	*W*[1]-hard	*FPT*
	$l + \Delta$	–	*FPT*	*FPT*	*FPT*	*FPT*
	tw	weakly *NP*-C	*PSPACE*-C	*PSPACE*-C	weakly *NP*-H	weakly *NP*-H
	$tw + \Delta$	*FPT*	*PSPACE*-C	*PSPACE*-C	*FPT*	*FPT*

PSPACE-complete even when played on $k \times n$ boards, where k is a constant. This is in contrast to the result of Ravikumar [12] that Peg Solitaire, a game *NP*-complete on $n \times n$ boards, is linear time solvable on $k \times n$ boards for any fixed k.

NCL is also has applications in showing the hardness of reconfiguration problems [10, 5, 2, 9]. For some reconfiguration problems, their hardness on planar graphs of low maximum degree is known by reduction from NCL [5, 9] while their hardness on bounded bandwidth graphs is known by reduction from H-Word Reconfiguration [14, 5, 11]. Our reduction, which combines techniques from Hearn and Demaine's constraint logic [8] and the reductions from H-Word Reconfiguration in [14, 11] unifies these results: we show that a number of reconfiguration problems (including Independent Set, Vertex Cover and Dominating Set) are *PSPACE*-complete on low-degree, planar graphs of bounded bandwidth. Previously, hardness was known only on graphs that *either* are planar and have low degree *or* have bounded bandwidth - we show that the problems remain hard even when both of these conditions hold simultaneously. Note that while a graph of bounded bandwidth also has bounded degree, the graphs created in these reductions have quite large bandwidth. The degree bounds we obtain are much tighter than what would be obtained from the bandwidth bound alone.

Our results concerning the hardness of constraint logic problems are summarized in Table 1. This table shows the parameterized complexity of Constraint Graph Satisfiability (CGS), unbounded configuration-to-edge (C2E) and configuration-to-configuration (C2C) variants of Nondeterministic Constraint Logic and their respective bounded counterparts (BC2E and BC2C) with respect to solution length (l), maximum degree (Δ) and treewidth (tw). If a traditional complexity class is listed this means that the problem is hard for this class even when restricted to instances where the parameter is at most a constant.

In this extended abstract, we discuss only the main result (concerning unbounded C2E and C2C NCL on bounded bandwidth graph) and its applications. For a discussion of the other results in Table 1 and for some omitted proofs, we refer to the full version of this paper [13].

2 Preliminaries

2.1 Constraint Logic

▶ **Definition 1** (Constraint Graph). A *constraint graph* is a graph with edge weights and vertex weights. A *legal configuration* for a constraint graph is an assignment of an orientation to each edge such that for each vertex, the total weight of the edges pointing into that vertex is at least that vertex' weight (its *minimum inflow*).

(a) OR vertex **(b)** AND vertex

Figure 1 The two vertex types from which a restricted constraint graph is constructed: (a) OR vertex and (b) AND vertex. Following the convention set in [6], as a mnemonic weight 2 edges are drawn blue (dark grey) and thick, while weight 1 edges are drawn red (light grey) and thinner.

A fundamental decision problem regarding constraint graphs is that of their satisfiability:

Constraint Graph Satisfiablility

Instance: A constraint graph G.

Question: Does G have a legal configuration?

Constraint Graph Satisfiability (CGS) is *NP*-complete [8].

An important problem regarding constraint graph configurations is whether they can be reconfigured into each other:

Nondeterministic Constraint Logic (C2C)

Instance: A constraint graph G and two legal configurations C_1, C_2 for G.

Question: Is there a sequence of legal configurations from C_1 to C_2, where every configuration is obtained from the previous configuration by changing the orientation of one edge?

This problem is called the configuration-to-configuration (C2C) variant of Nondeterministic Constraint Logic . It is *PSPACE*-complete [8]. The configuration-to-edge variant (C2E) is also *PSPACE*-complete [8]:

Nondeterministic Constraint Logic (C2E)

Instance: A constraint graph G, a target edge e from G and an initial legal configuration C_1 for G.

Question: Is there a sequence of legal configurations, starting with C_1, where every configuration is obtained from the previous by changing the orientation of one edge, so that eventually e is reversed?

For the C2C and C2E problems, bC2C and bC2E denote their bounded variants which ask whether there exists a reconfiguration sequence in which each edge is reversed at most once. These problems are *NP*-complete [8].

Hearn and Demaine [8] consider a restricted subset of constraint graphs, which are planar and constructed using only two specific types of vertices: AND and OR vertices (Figure 1).

The OR vertex has minimum inflow 2 and three incident weight 2 edges. Its inflow constraint is thus satisfied if and only if at least one of its incident edges is directed inwards (resembling an OR logic gate). The AND vertex has minimum inflow 2, two incident weight 1 edges and one incident weight 2 edge. Its constraint is thus satisfied if and only if both weight 1 edges are directed inwards or the weight 2 edge is directed inwards (resembling an AND logic gate). Both C2C and C2E NCL remain *PSPACE*-complete under these restrictions.

(a) Constrained blue-edge terminator **(b)** Free red-edge terminator

Figure 2 Gadgets for terminating loose edges. The constrained blue-edge terminator (a) forces the blue edge A to point into the gadget, while the free red-edge terminator (b) allows the red edge A to point out of the gadget.

2.2 Constraint Logic Gadgets

Some constructions in this paper use existing gadgets (such as crossover) for NCL reductions due to Hearn and Demaine [8]. For the purpose of being self-contained, we reproduce these gadgets here and state (without proof) their functionality.

Edge Terminators. The constrained blue-edge (Figure 2a) terminator allows us to have a loose blue edge that is forced to point outwards, effectively removing the edge from the graph while still meeting the requirement that the graph is built from only AND and OR vertices. The free red-edge terminator (Figure 2b) allows us to have a loose red edge whose orientation can be freely chosen, effectively decreasing the minimum inflow of the vertex to which it is incident by one.

Red-blue Conversion. It is useful to be able to convert a blue edge to a red edge, i.e. we require a gadget which has a blue edge that can (be reconfigured to) point outwards if and only if its red edge is pointing inwards and vice-versa. Hearn and Demaine [8] provide a construction that allows red-blue conversion in pairs, but also note a simpler construction is possible: an AND vertex, with one of its red edges attached to a free red-edge terminator (Figure 2b) can serve as a red-blue conversion gadget.

We also use the crossover gadget due to Hearn and Demaine [8]. We do not reproduce this gadget here, as it is sufficient to know that given a constraint graph we can eliminate any crossings using the crossover gadget, which can be constructed using only AND and OR vertices. For details of the crossover gadget, we refer to [13, 8].

2.3 H-Word Reconfiguration

To show hardness of NCL on bounded bandwidth graphs, we reduce from the H-Word Reconfiguration problem, introduced in [14].

▶ **Definition 2** (H-word)**.** Let $H = (\Sigma, E)$ where Σ is an alphabet and $E \subseteq \Sigma \times \Sigma$ a relation. An *H-word* is a word over Σ such that every pair of consecutive characters (a, b) is an element of E.

H-Word Reconfiguration

Instance: Two H-words W_s, W_g of equal length

Question: Is there a sequence of H-words $W_1, \ldots, W_n$, so that every pair of consecutive words W_i, W_{i+1} can be obtained from each other by changing one character to another and $W_1 = W_s, W_n = W_g$?

▶ **Theorem 3** (Wrochna [14]). *There exists an H such that H-Word Reconfiguration is PSPACE-complete.*

2.4 Bandwidth and Cutwidth

The main result concerns graphs of bounded bandwidth. The proof of Theorem 11 uses the notion of cutwidth.

▶ **Definition 4** (Bandwidth [1]). Let $G = (V, E)$ be a graph and define a one-to-one correspondence $f : V \to \{1, \ldots, |V|\}$. The *bandwidth* of a graph is the minimum over all such correspondences of $\max_{(u,v)\in E} |f(u) - f(v)|$.

▶ **Definition 5** (Cutwidth [1]). Let $G = (V, E)$ be a graph and define a one-to-one correspondence $f : V \to \{1, \ldots, |V|\}$. The *cutwidth* of a graph is the minimum over all such correspondences of $\max_{w\in V} |\{(u, v) \in E | f(u) \leq f(w) < f(v)\}|$.

3 Hardness of Nondeterministic Constraint Logic on Bounded Bandwidth Graphs

In this section we prove the main result, namely that restricted Nondeterministic Constraint Logic remains *PSPACE*-complete even when restricted to graphs of bounded bandwidth (which is a subclass of graphs of bounded treewidth).

To show *PSPACE*-completeness, we reduce from H-Word Reconfiguration. The bandwidth of the constraint graph created in the reduction will depend only on the size of H. Since H-Word Reconfiguration is *PSPACE*-complete for a fixed (finite) H, we obtain a constant bound on the bandwidth.

▶ **Theorem 6.** *There exists a constant c, such that C2C Nondeterministic Constraint Logic is PSPACE-complete on planar constraint graphs of bandwidth at most c that consist of only AND and OR vertices.*

Proof. Let $H = (\Sigma, E)$ be so that H-Word Reconfiguration is *PSPACE*-complete. We provide a reduction from H-Word Reconfiguration to (unrestricted) NCL and then show how to adapt this reduction to work for restricted NCL. Let W_s, W_g be an instance of H-Word Reconfiguration and let n denote the length of W_s. In the following, all vertices are given minimum inflow 2.

We create a matrix of vertices $X_{i,j}$ for $i \in \{1, \ldots, n\}, j \in \Sigma$, the *character vertices*. The orientation of edges incident to $X_{i,j}$ will correspond to whether the character at position i in the word is character j. For every row i of this matrix we create a *universal vertex* U_i and for every $j \in \Sigma$ we create a blue (weight 2) edge connecting U_i and $X_{i,j}$.

In each row $i \in 1, \ldots, n-1$ and for all pairs $(A, B) \notin E$, we create a *relation vertex* $\Delta_{i,A,B}$. We create a red (weight 1) edge connected to $X_{i,A}$ and pass it through a red-blue conversion gadget and connect the blue edge leaving the conversion gadget to $\Delta_{i,A,B}$. We mirror this construction, creating a red edge connected to $X_{i+1,B}$, converting it to blue, and connecting it to $\Delta_{i,A,B}$.

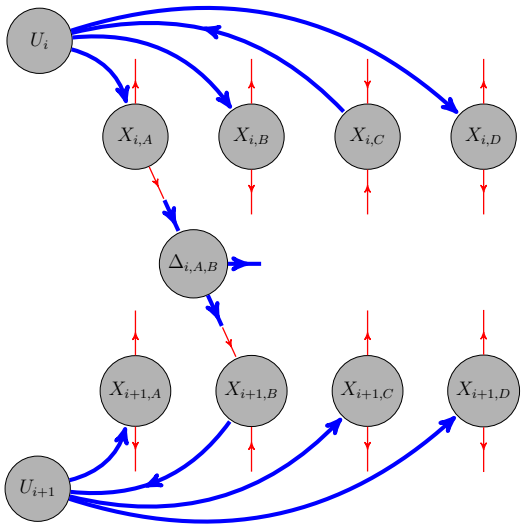


Figure 3 Slice of two rows from the matrix of vertices (over an alphabet of $\Sigma = \{A, B, C, D\}$), and the construction for enforcing non-adjacency of A and B.

Finally, we connect one additional blue edge to $\Delta_{i,A,B}$ and connect it to a constrained blue-edge terminator. This edge serves no purpose (its inflow can never count towards $\Delta_{i,A,B}$) but to make $\Delta_{i,A,B}$ an OR vertex as claimed.

A single instance of this construction is shown in Figure 3, which depicts a slice of two rows from the matrix of vertices. Having created an instance of this construction for (A, B), we might need to create an instance for (A, C). Rather than attaching another red edge to $X_{i,A}$, we instead split the existing red edge leaving $X_{i,A}$ by connecting it to a red-blue conversion gadget, and attaching the resulting blue edge to an AND vertex. The edges leaving the AND vertex may be oriented outward if and only if the red edge leaving $X_{i,A}$ is oriented into the AND vertex, effectively splitting it. We then convert the two red edges of the AND vertex to blue and attach them to $\Delta_{i,A,B}$ and $\Delta_{i,A,C}$ as before. To create further copies of this construction we can repeat the splitting process, so that $X_{i,A}$ eventually has two incident red edges, one connecting to gadgets in row $i-1$ and one connecting to row $i+1$ (the excess red edges in rows 1 and n may instead be connected to a free red edge terminator).

We define a character j to be *in* the word at position i if the edge $(U_i, X_{i,j})$ is oriented towards U_i.

▶ **Claim.** *In any legal configuration, at least one character is in the word at each position.*

Proof. U_i has minimum inflow 2, so at least one of its incident edges $(U_i, X_{i,j})$ must be oriented towards it and hence j is in the word at position i. ◀

▶ **Claim.** *In any legal configuration, if $(A, B) \notin E$ then A can be in the word at position i only if B is not in the word at position $i+1$.*

Proof. If A is in the word at position i, then $X_{i,A}$ is not receiving inflow from U_i, so both of the red edges incident to $X_{i,A}$ most point in towards $X_{i,A}$. On the path from $X_{i,A}$ through $\Delta_{i,A,B}$ to $X_{i+1,B}$ we will first pass through a red-blue conversion gadget. Since the red edge is oriented out of the conversion gadget (and towards $X_{i,A}$), the blue edge must be oriented

into the conversion gadget. We may then encounter several AND vertices (that are used for the "splitting" of red edges incident to $X_{i,A}$), note that since the blue edge is oriented towards the conversion gadget and away from the AND vertex, both red edges must be oriented into the AND vertex. This means that when we encounter another red-blue conversion gadget its blue edge must be oriented into the AND vertex and in turn the red edge incident to the conversion gadget has to be oriented into the conversion gadget. Ultimately, when we arrive at the vertex $\Delta_{i,A,B}$ we find that one of its edges must be oriented out of it (to point into the previously described AND vertices and help satisfy the minimum inflow of $X_{i,A}$). Suppose by contradiction that B is in the word at position $i+1$. By a similar argument, we find that the second edge incident to $\Delta_{i,A,B}$ also has to be oriented out of it. This is impossible, since the third blue edge incident to $\Delta_{i,A,B}$ is also oriented out of it (since that edge is attached to a constrained blue-edge terminator gadget) and thus its minimum inflow constraint is violated. Therefore B can not be in the word at position $i+1$. ◀

Note that this claim implies that if for each position we pick *some* character that is in the position at that word, we end up with a valid H-Word. We say that a configuration for the constraint graph *encodes* a word W if each of that word's characters is in the word at the appropriate position.

▶ **Claim.** *If a legal configuration for the constraint graph encoding W_s may be reconfigured into a legal configuration encoding only W_g (i.e. the configuration encodes no other word), then W_s may be reconfigured into W_g.*

Proof. Suppose we have a reconfiguration sequence of legal configurations $C_1, \ldots, C_m$ so that C_1 encodes W_s and C_m encodes only W_g. We define a reconfiguration sequence of words $W_1, \ldots, W_m$ which has the property that C_i encodes W_i. Let $W_1 = W_s$, we recursively define $W_i, 1 < i \leq m$ as follows: Since C_{i+1} differs from C_i in the orientation of only one edge, the set of words encoded by C_{i+1} differs only from the set of words encoded by C_i by making one character at a position allowed (in the word) or disallowed (was in the word in C_i but not in the word in C_{i+1}). If a character at a position becomes allowed in C_{i+1}, let $W_{i+1} = W_i$ (since C_i encodes W_i, C_{i+1} also encodes W_i). If a character at a position becomes disallowed (i.e. is no longer in the word), we obtain W_{i+1} from W_i by changing the character at that position to some allowed character (of which there is at least one). Following these steps, since the final configuration encodes only W_g, we obtain a reconfiguration sequence from W_s to W_g as claimed. ◀

Note that we may have multiple choices for W_i because C_i can encode more than one word. It suffices to simply pick one of the words it encodes, while ensuring it differs in at most one character from the previous and next.

▶ **Claim.** *Given a H-word W of length n, there exists a legal configuration for the constraint graph encoding only W.*

Proof. Pick the orientation of edge $(U_i, X_{i,j})$ to be towards U_i if the character in W at position i is j, and towards $X_{i,j}$ otherwise. Clearly this constraint graph encodes only W. This configuration can be extended to a legal configuration for the remaining (relation) vertices by noticing the following: if there is a relation vertex $\Delta_{i,A,B}$ then either A is not in the word at position i or B is not in the word at position j. Suppose w.l.g. that B is not in the word, then $X_{i+1,B}$ is receiving inflow from U_{i+1} and hence its incident red edges may be pointing outwards, which (after passing through the red-blue conversion gadgets and

splitting AND vertices as described before) allows us to satisfy the inflow requirement of $\Delta_{i,A,B}$. ◀

▶ **Claim.** *If two H-Words W_1, W_2 differ by only one character, then a constraint graph G encoding only W_1 can be reconfigured into one encoding only W_2.*

Proof. Suppose that W_1 and W_2 differ by changing the character at position i from A to B. We may first reconfigure G from encoding only W_1 to encoding both W_1 and W_2, and then reconfigure it to encode only W_2. This temporarily increases the inflow U_i receives from 2 (receiving inflow only from $X_{i,A}$) to 4 (receiving inflow from both $X_{i,A}$ and $X_{i,B}$) back to 2 (receiving inflow from only $X_{i,A}$). It may be required to reorient some edges to satisfy the inflows of the relation vertices, but this is always possible because neither A nor B conflicts with any preceding or succeeding characters. ◀

Note that this claim implies that if W_s can be reconfigured into W_g, then a constraint graph encoding only W_s can be reconfigured into one encoding only W_g. This completes the reduction.

All that remains to show is that this graph can be adapted so that it only uses AND and OR vertices and becomes planar, and that the resulting graph has bounded bandwidth. The only vertices that are not already AND or OR vertices are the universal vertices U_i. Note that taking a single OR vertex (that has 3 incident edges at least one of which has to be oriented inwards) we can attach another OR vertex to one of its edges to obtain a structure that has 4 external edges, at least one of which has to be oriented inwards. Repeating this procedure n times we obtain a tree of OR vertices with $n+3$ external edges, at least one of which has to be oriented inwards, which can replace the universal vertex.

To make the graph planar, we can use Hearn and Demaine's crossover gadget [8, 13]. While this may increase the graph's bandwidth, the following argument that it remains bounded by a constant:

We create a number of bags $B_1, \dots B_{n-1}$ that are subsets of vertices, with the property that the size of each bag depends only on H (which is fixed) and that edges connect only vertices that are both in the same bag, or a vertex in bag B_i with a vertex in bag B_{i+1}. This is achieved by taking in bag i the vertices $\{X_{i,j} : j \in \Sigma\}, \{\Delta_{i,A,B}\,(A,B) \notin E\}, \{X_{i+1,j} : j \in \Sigma\}$, the vertices in gadgets between them (i.e. the red-blue conversion gadgets and AND vertices used in the splitting process) and the construction replacing the universal vertices U_i and U_{i+1}. This shows the bandwidth of the graph is bounded by a constant c, since we can order the vertices so that a vertex in B_i precedes a vertex in B_{i+1} (and pick an arbitrary order for the vertices in the same bag).

We have thus shown how to reduce H-Word Reconfiguration to NCL, shown how to adapt our reduction to use only AND and OR vertices and make the resulting graph planar and that the resulting graph has bounded bandwidth and have thus shown Theorem 6. ◀

Theorem 6 also holds for C2E Nondeterministic Constraint Logic:

▶ **Theorem 7.** *There is a constant c, such that C2E* Nondeterministic Constraint Logic *is PSPACE-complete, even on planar constraint graphs of treewidth (bandwidth) at most c that use only AND and OR vertices.*

Proof. H-Word Reconfiguration remains *PSPACE*-complete, even when instead of requiring that one H-word is reconfigured in to another, we ask whether it is possible to reconfigure a given H-word so that a given character appears at a specific position (without requiring anything regarding the remaining characters in the word). This can be seen by

examining the proof in [14], noting that we may modify the Turing machine to, upon reaching an accepting state, move the head to the start of the tape and write a specific symbol there. The proof of Theorem 6 can then easily be adapted to work for C2E NCL (since a character appearing at a specific position corresponds to reversing a specific edge). ◀

We note that Theorems 6 and 7 may be further strengthened by requiring that all OR vertices in the graph are *protected*, i.e., in any legal configuration at least one of its incident edges is directed outwards. Hearn and Demaine [8] show how to construct an OR vertex using only AND vertices and protected OR vertices.

4 Applications

As a result of our hardness proof for NCL, we (nearly) automatically obtain tighter bounds for other problems that reduce from NCL in their hardness proofs. If the reduction showing *PSPACE*-hardness for a problem is bandwidth-preserving in the sense that the bandwidth of the resulting graph only depends on the bandwidth of the original constraint graph, then that problem remains *PSPACE*-hard even when limited to instances of bounded bandwidth. Since reductions from NCL usually work by locally replacing AND and OR vertices in the graph with gadgets that simulate their functionality, such reductions are often bandwidth-preserving.

Independent Set Reconfiguration (IS-R)

Instance: Graph $G = (V, E)$, independent sets $S_s, S_g \subseteq V$ of G, integer threshold k.

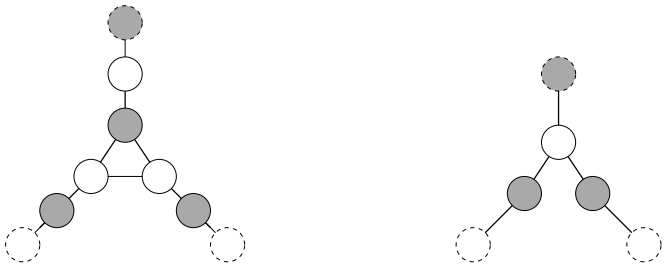
Question: Is there a sequence of independent sets $S_1, \ldots, S_n$, so that for all i, S_{i+1} is obtained from S_i by the addition or removal of one vertex, $|S_i| \geq k$ and $S_1 = S_s, S_n = S_g$?

This is the *token addition-and-removal (TAR)* version of IS-R. It is also possible to define a *token jumping (TJ)* variant (in which we obtain one independent set from the next by removing and immediately replacing a vertex) and a *token sliding* variant (in which when we remove a vertex, we must replace it with an incident vertex, i.e. "sliding" the token/vertex along an edge).

▶ **Theorem 8.** *TAR, TJ and TS versions of* Independent Set Reconfiguration *are PSPACE-complete on planar, maximum degree 3 graphs of bounded bandwidth.*

Proof. Hearn and Demaine [7] provide a reduction from IS-R to TJ and TS versions of IS-R. The gadgets used in their reduction are reproduced in Figure 4. Since the reduction works by replacing every vertex with a construction of at most 6 other vertices, the bandwidth of the graph created in the reduction is at most 6 times the bandwidth of the original constraint graph. The theorem thus follows immediately from our hardness result and Hearn and Demaine's reduction [7] and the fact that the TAR version is equivalent to TJ [10]. ◀

Similarly to Independent Set Reconfiguration, we can define reconfiguration versions of Vertex Cover (VC-R), Feedback Vertex Set (FVS-R), Induced Forest (IF-R), Odd Cycle Transversal (OCT-R) and Induced Bipartite Subgraph (IBS-R). Note that for IS-R, IF-R, IBS-R the size of the vertex subset is never allowed to drop below the threshold, while for VC-R, FVS-R and OCT-R the size is never allowed to exceed the threshold. These problems are *PSPACE*-complete on bounded bandwidth graphs by reduction from H-Word Reconfiguration [11]. We strengthen this result, noting that our proof follows the same reasoning as in [11]:



(a) OR vertex **(b)** AND vertex

Figure 4 Gadgets used in the reduction from IS-R to NCL. Dashed vertices represent vertices that are part not of the gadget itself, but of other gadgets. The dark gray vertices represent an example independent set.

▶ **Theorem 9.** *TAR, TJ and TS versions of* Independent Set *(IS-R),* Vertex Cover *(VC-R),* Feedback Vertex Set *(FVS-R),* Induced Forest *(IF-R),* Odd Cycle Transversal *(OCT-R) and* Induced Bipartite Subgraph *(IBS-R) are PSPACE-complete on planar graphs of bounded bandwidth and low maximum degree.*

Proof. By Theorem 8, IS-R is *PSPACE*-complete on planar graphs of bounded bandwidth and maximum degree 3. Since an independent set is the complement of a vertex cover, the theorem holds for VC-R on maximum degree 3 graphs.

By replacing every edge in the graph by a triangle, we can reduce VC-R to FVS-R (since picking at least one vertex of every edge is equivalent to picking at least one vertex of every triangle), thus showing the theorem for FVS-R on maximum degree 6 graphs. We note that in such a graph a feedback vertex set is also an odd cycle transversal, thus also showing the theorem for OCT-R on maximum degree 6 graphs.

Finally, the theorem holds for IF-R and IBS-R on maximum degree 6 graphs by considering complements of solutions for FVS-R and OCT-R. ◀

Another related reconfiguration problem is that of Dominating Set Reconfiguration (DS-R). In [5], the authors show that DS-R is *PSPACE*-complete on planar graphs of maximum degree six by reduction from NCL and *PSPACE*-complete on graphs of bounded bandwidth by a reduction from VC-R. The following theorem that unifies these results follows immediately from our improved result concerning VC-R:

▶ **Theorem 10.** *The TAR version of* Dominating Set Reconfiguration *is PSPACE-complete, even on planar, maximum degree six graphs of bounded bandwidth.*

The puzzle game Rush Hour, in which the player moves cars horizontally and vertically with the goal to free a specific car from the board, is *PSPACE*-complete [4] when played on an $n \times n$ board. Another consequence of our result is that Rush Hour is *PSPACE*-complete even on rectangular boards where one of the dimensions of the board is constant:

▶ **Theorem 11.** *There exists a constant k, so that Rush Hour is PSPACE-complete when played on boards of size $k \times n$.*

Proof. Hearn and Demaine [7] provide a reduction from restricted NCL, showing how to construct AND and OR vertices as gadgets in Rush Hour and how to connect them together using straight line and turn gadgets. Given a planar constraint graph, it can be drawn in the grid (with vertices on points of the grid and edges running along the lines of the grid), after

which vertices can be replaced by their appropriate gadgets and edges with the necessary line and turn gadgets. However, starting with a graph of bounded bandwidth does not immediately give a suitable drawing in a grid of which one of the dimensions is bounded (from which the theorem would follow, since all of the gadgets have constant size).

Given a restricted NCL graph of bounded bandwidth, we note that this graph also has bounded cutwidth [1]. Given such a graph with n vertices and cutwidth at most c, it is possible to arrange it in a $(c+1) \times 3n$ grid, so that vertices of the graph are mapped to vertices of the grid, the edges of the graph run along edges of the grid, and no two edges or vertices get mapped to the same edge or vertex in the grid. This is achieved by placing all of the vertices of the graph in a single column (noting that 3 rows are required for each vertex since it has 3 incident edges) and (due to the graph having cutwidth c) the edges can be placed in the remaining c columns (placing each edge in a distinct column). However, even when starting with a planar NCL graph, the resulting embedding may have crossings. These can be eliminated using the crossover gadget, noting that since the crossover gadget has constant size, the resulting graph can still be drawn in a $O(c) \times O(n)$ grid. We can now use the existing gadgets from [7] to finish the reduction, showing Rush Hour *PSPACE*-complete on boards of which one of the dimensions is bounded by a constant. ◀

Note that this result is likely to carry over to show other board games *PSPACE*-complete on $n \times k$ boards, such as Sokoban or Plank Puzzles, which also reduce from NCL in their hardness proofs [8].

This result is in contrast to [12], where it is shown that Peg Solitaire, when played on $k \times n$ boards, is linear time solvable for any fixed k. An important distinction here is that the length of a solution in a Peg Solitaire game is *bounded* by the number of pegs (since every move removes one peg from play) whereas Rush Hour games are *unbounded*: any length move sequence is possible, though obviously after an exponential number of moves, positions would be repeated.

5 Conclusions

We have studied the parameterized complexity of constraint logic problems with regards to (combinations of) solution length, maximum degree and treewidth as parameters. As a main result, we showed that RESTRICTED NONDETERMINISTIC CONSTRAINT LOGIC remains *PSPACE*-complete on graphs of bounded bandwidth, strengthening Hearn and Demaine's framework [8].

By combining Wrochna's proof [14] and Hearn and Demaine's [8] constraint logic techniques, we have managed to get the best of both worlds: for several reconfiguration problems, we showed hardness for graphs that are not only planar and have low maximum degree, but that also have bounded bandwidth. This not only strengthens Wrochna's results, but also makes it easier to prove hardness for reconfiguration on bounded bandwidth graphs by merging H-WORD RECONFIGURATION into the more convenient NCL framework.

Note that the constant c in Theorem 6 has not been calculated precisely, but an informal analysis of Wrochna's proof [14] and our reduction suggests it is very large (growing with the 12^{th} power of the original instance size). This raises two open questions: one is to determine a tighter bound on the value of c, and the other is to determine whether efficient algorithms exist for solving reconfiguration problems when the graph's bandwidth is bounded by more practical values. Some progress in this direction has already been made [3].

We have studied the parameterized complexity of CGS and NCL. Hearn and Demaine [8] have defined several other problems related to constraint graphs, including two player

and multiple player constraint graph games, complete for classes such as *EXPTIME* and *NEXPTIME*. Studying these problems in a parameterized setting gives rise to several interesting open problems.

Acknowledgement. I thank my advisor, Hans Bodlaender, for his guidance and useful comments and suggestions.

References

1. Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical computer science*, 209(1):1–45, 1998.
2. Paul Bonsma and Luis Cereceda. Finding paths between graph colourings: Pspace-completeness and superpolynomial distances. *Theoretical Computer Science*, 410(50):5215–5226, 2009.
3. 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 *Algorithms and Computation*, pages 389–400. Springer, 2014.
4. Gary William Flake and Eric B. Baum. Rush hour is pspace-complete, or "why you should generously tip parking lot attendants". *Theoretical Computer Science*, 270(1):895–911, 2002.
5. Arash Haddadan, Takehiro Ito, Amer E. Mouawad, Naomi Nishimura, Hirotaka Ono, Akira Suzuki, and Youcef Tebbal. The complexity of dominating set reconfiguration. *arXiv preprint arXiv:1503.00833*, 2015.
6. Robert A. Hearn and Erik D. Demaine. The nondeterministic constraint logic model of computation: Reductions and applications. In *Automata, Languages and Programming*, pages 401–413. Springer, 2002.
7. Robert A. Hearn and Erik D. Demaine. Pspace-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science*, 343(1):72–96, 2005.
8. Robert A. Hearn and Erik D. Demaine. *Games, puzzles, and computation*. CRC Press, 2009.
9. 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. In *Algorithms and Computation*, pages 28–39. Springer, 2008.
10. Marcin Kamiński, Paul Medvedev, and Martin Milanič. Complexity of independent set reconfigurability problems. *Theoretical computer science*, 439:9–15, 2012.
11. Amer E. Mouawad, Naomi Nishimura, Venkatesh Raman, and Marcin Wrochna. Reconfiguration over tree decompositions. In *Parameterized and Exact Computation*, pages 246–257. Springer, 2014.
12. Bala Ravikumar. Peg-solitaire, string rewriting systems and finite automata. In *Algorithms and Computation*, pages 233–242. Springer, 1997.
13. Tom C. van der Zanden. Parameterized complexity of graph constraint logic. *arXiv preprint:1509.02683*, 2015.
14. Marcin Wrochna. Reconfiguration in bounded bandwidth and treedepth. *arXiv preprint arXiv:1405.0847*, 2014.

Complexity and Approximability of Parameterized MAX-CSPs

Holger Dell[1], Eun Jung Kim[2], Michael Lampis[3], Valia Mitsou[4], and Tobias Mömke[5]

1 **Saarland University and Cluster of Excellence, Saarbrücken, Germany**
Simons Institute for the Theory of Computing, Berkeley, US
hdell@mmci.uni-saarland.de
2 **Université Paris Dauphine, France**
Simons Institute for the Theory of Computing, Berkeley, US
eunjungkim78@gmail.com
3 **Université Paris Dauphine, France**
michail.lampis@dauphine.fr
4 **SZTAKI, Hungarian Academy of Sciences, Budapest, Hungary***
vmitsou@gradcenter.cuny.edu
5 **Saarland University, Saarbrücken, Germany†**
moemke@cs.uni-saarland.de

Abstract

We study the optimization version of constraint satisfaction problems (Max-CSPs) in the framework of parameterized complexity; the goal is to compute the maximum fraction of constraints that can be satisfied simultaneously. In standard CSPs, we want to decide whether this fraction equals one. The parameters we investigate are structural measures, such as the treewidth or the clique-width of the variable–constraint incidence graph of the CSP instance.

We consider Max-CSPs with the constraint types AND, OR, PARITY, and MAJORITY, and with various parameters k. We attempt to fully classify them into the following three cases:

1. The exact optimum can be computed in FPT-time.
2. It is W[1]-hard to compute the exact optimum, but there is a randomized FPT approximation scheme (FPT-AS), which computes a $(1-\epsilon)$-approximation in time $f(k,\epsilon)\cdot \mathrm{poly}(n)$.
3. There is no FPT-AS unless FPT=W[1].

For the corresponding standard CSPs, we establish FPT vs. W[1]-hardness results.

1998 ACM Subject Classification F.2.0 [Analysis of Algorithms and Problem Complexity] General

Keywords and phrases Approximation, Structural Parameters, Constraint Satisfaction

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.294

1 Introduction

Constraint Satisfaction Problems (CSPs) play a central role in almost all branches of theoretical computer science. Starting from CNF-SAT, the prototypical NP-complete problem, the computational complexity of CSPs has been widely studied from various points of view.

* Supported by ERC Starting Grant PARAMTIGHT (No. 280152)
† This research is supported by Deutsche Forschungsgemeinschaft grant BL511/10-1

In this paper we focus on two aspects of CSP complexity which, though extremely well-investigated, have mostly been considered separately so far in the literature: parameterized complexity and approximability. We study four standard predicates and contribute some of the first results indicating that the point of view of approximability considerably enriches the parameterized complexity landscape of CSPs.

1.1 Parameterized CSPs

The vast majority of interesting CSPs are NP-hard [21, 12]. This has motivated the study of such problems from a parameterized complexity point of view, and indeed this topic has attracted considerable attention in the literature [9, 26, 6, 18, 8, 25]. We refer the reader to [20] where an extensive classification of CSP problems for a large range of parameters is given. In this paper we focus on *structurally* parameterized CSPs, that is, we consider CSPs where the parameter is some measure of the structure of the input instance. The central idea behind this approach is to represent the structure of the CSP using a (hyper-)graph and leverage the powerful tools commonly applied to parameterized graph problems (such as tree decompositions) to solve the CSP.

The typical goal of this line of research is to find the most general parameterization of a CSP that still remains fixed-parameter tractable (FPT). To give a concrete example for a very well-known CSP, CNF-SAT is FPT when parameterized by the treewidth of its incidence graph[1] [23] but it is W-hard for more general parameters such as clique-width [16], or even the more restricted modular treewidth [17]. General (boolean) CSP on the other hand, where the description of each constraint is part of the input is known to be a harder problem: it is already W[1]-hard parameterized by the incidence treewidth, but FPT parameterized by the treewidth of the primal graph [24]. Thus, parameterized investigations aim to locate the boundary where a CSP jumps from being FPT to being W-hard. It is of course a natural question how we can deal with the W-hard cases of a CSP once they are identified.

1.2 Approximation

CSPs also play a central role in the theory of (polynomial-time) approximation algorithms [27, 13, 2]. In this context we typically consider a CSP as an optimization problem (MAX-CSP) where the goal is to find an assignment to the variables that satisfies as many of the constraints as possible. Unfortunately, essentially all non-trivial CSPs are hard to approximate (APX-hard) from this point of view [4, 12], even those where deciding if an assignment can satisfy all constraints is in P (e.g. 2CNF-SAT or Horn SAT). Thus, research in this area typically focuses on discovering exactly the best approximation ratio that can be achieved in polynomial time. Amazingly, for many natural CSPs this happens to be exactly the ratio achieved by a completely random assignment [11]. This motivates the question of whether we can find natural cases where non-trivial efficient approximations are possible.

1.3 Results

In this paper we consider four different types of CSPs where the constraints are respectively OR, AND, PARITY and MAJORITY functions. Our approach follows, for the most part, the standard parameterized complexity script: we consider the input instance's incidence graph and try to determine the complexity of the CSP when parameterized by various graph

[1] See the next section for a definition of incidence graphs

widths. The new ingredient in our approach is that, in addition to trying to determine which parameters make a CSP FPT or W-hard, we also ask if the optimization versions of W-hard cases can be well-approximated. We believe that this is a question of special interest since, as it turns out, there are CSPs for which W-hardness can be (almost) circumvented using approximation, and others which are inapproximable.

More specifically, our results are as follows: for OR constraints, which corresponds to the standard CNF-SAT (MAX-CNF-SAT) problem, we present a new hardness proof establishing that deciding a formula's satisfiability is W-hard even if parameterized by the incidence graph's neighborhood diversity. Neighborhood diversity is a parameter much more restricted than modular treewidth (already a restriction of clique-width) [14], for which the strongest previously known W[1]-hardness result was known [17]. We complement this negative result with a strong positive approximation result: there exists a randomized FPT Approximation Scheme (FPT-AS)[2] for MAX-CNF-SAT parameterized by clique-width, that is, an algorithm which for all $\epsilon > 0$ runs in time $f(k, \epsilon)n^{O(1)}$ and returns an assignment satisfying $(1-\epsilon)$OPT clauses. Thus, even though we establish that solving CNF-SAT exactly is W-hard even for extremely restricted dense graph parameters, MAX-CNF-SAT is well-approximable even in the quite general case of clique-width. To the best of our knowledge, this is the first approximation result of this type for a W-hard MAX-CSP problem.

Recalling that MAX-CNF-SAT is FPT parameterized by the treewidth of the incidence graph, we consider other problems for which the jump from treewidth to clique-width could have interesting complexity consequences. We show that MAX-DNF-SAT and MAX-PARITY, which are FPT parameterized by treewidth, exhibit two wildly different behaviors. On the one hand, the problem of maximizing the largest possible number of satisfied PARITY constraints remains FPT even for dense parameters such as clique-width. On the other hand, by modifying our reduction for CNF-SAT, we are able to show not only that maximizing the number of satisfied AND constraints is W[1]-hard parameterized by neighborhood diversity, but also that this problem cannot even admit an FPT-AS (like MAX-CNF-SAT), unless W[1]=FPT. We recall that PARITY and AND constraints are similar in other aspects: for example, for both we can decide in polynomial time if an assignment satisfying all constraints exists.

Finally, we consider CSPs with MAJORITY constraints, that is, constraints which are satisfied if at least half their literals are true. We give a reduction establishing that this is an interesting case of a natural constraint type for which deciding satisfiability is already W[1]-hard parameterized by treewidth (we actually show W[1]-hardness for the more restricted case of incidence feedback vertex set) and by neighborhood diversity. We complement this negative result with two algorithmic results: first, we show that the corresponding MAX-CSP is FPT parameterized by incidence vertex cover. Then, we use this algorithm as a sub-routine to obtain an FPT-AS for incidence feedback vertex set. Both of these algorithmic results also apply to the more general case of THRESHOLD constraints. We leave it as an interesting open problem to examine if the approximation algorithm for feedback vertex set can be extended to treewidth.

2 Preliminaries

A Boolean CSP ψ is defined as a set $\{C_1, \ldots, C_m\}$ of m constraints over a set $X(\psi) = \{x_1, \ldots, x_n\}$ of n variables and their negations. Each constraint C_i is regarded as a function

[2] We follow the standard definition of FPT-AS given in [15].

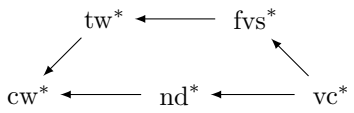


Figure 1 The structural parameters we study and their relationships. For example, the arrow between cw* and tw* means that if the treewidth is bounded, then the clique-width is bounded as well — more precisely, there is a monotone computable function $f : \mathbf{N} \to \mathbf{N}$ so that $\text{cw}^* \leq f(\text{tw}^*)$. On the other hand, tw*and nd*as well as fvs*and nd*cannot be bounded by each other in general.

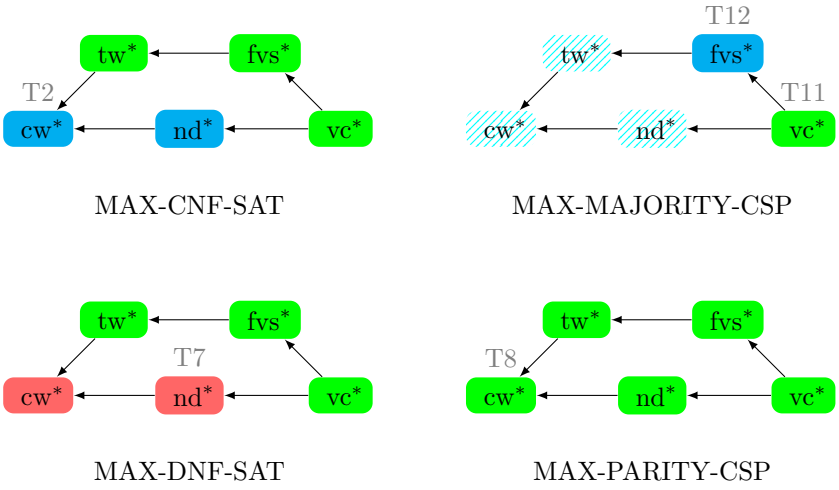


Figure 2 The parameterized complexity status of MAX-CSP problems. The *gray* labels above the boxes indicate the theorem in which we establish the result; previously known results are displayed without reference. *Red* means that the problem is W[1]-hard to compute exactly, and there is no FPT-AS unless FPT = W[1]. *Blue* means that the problem is W[1]-hard to compute exactly, and there is an FPT-AS. *Green* means that the problem is FPT to compute exactly. The *blue/white* stripes mean that it's W[1]-hard to compute exactly, and it's open whether there is an FPT-AS.

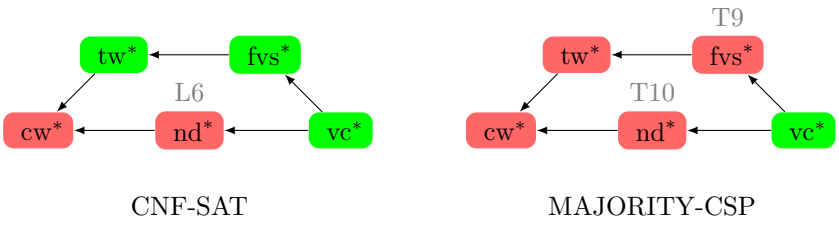


Figure 3 The parameterized complexity status of CNF-SAT and MAJORITY-CSP. Recall that DNF-SAT and PARITY-CSP are polynomial-time computable. *Red* means that the problem is W[1]-hard and *green* means that the problem is FPT.

of literals (positive or negative appearances of variables) mapped to the set $\{0,1\}$, where literals can take the values 0 or 1. Furthermore, we define $|C_j|$ to denote the arity of constraint C_j (the number of literals that occur in C) and $|\psi| = m$ the number of constraints in ψ. For simplicity, we also write $l_i \in C_j$ for a literal l_i and a constraint C_j if l_i appears in C_j.

We will be dealing with Boolean constraint satisfaction problems for four well-studied Boolean functions: OR constraints, AND constraints, PARITY (or XOR) constraints and MAJORITY constraints. We say that an assignment $t : X \to \{0,1\}$ satisfies a constraint C of

- OR, if $\exists l_i \in C$, $t(l_i) = 1$;
- AND, if $\forall l_i \in C$, $t(l_i) = 1$;
- PARITY, if it satisfies some equation $\Sigma_{l_i \in C} t(l_i) = b$ (for $b \in \{0,1\}$) modulo 2;
- MAJORITY, if at least $\lceil |C|/2 \rceil$ literals in C are set to 1. More generally, we may consider THRESHOLD constraints, where a certain threshold number of literals must be set to true to satisfy the constraint.

Let $\mathrm{occ}(\psi) = \sum_{C \in \psi} |C|$ be the total number of variable occurrences in ψ, that is, the total size of the formula. For a variable x, we write ψ_x for the set of all constraints $C \in \psi$ where x occurs either positive or negative; for the functions we consider without loss of generality, no clause contains both literals. Thus, the total number of occurrences of a variable x is equal to $|\psi_x|$.

We are dealing also with MAX-CSPs, where given a set of constraints ψ, we are interested in finding an assignment to the variables that maximizes the number of satisfied constraints. The natural decision version of this problem is, given a target k, decide whether there exists an assignment that satisfies at least k constraints. Clearly, the problem where we want to decide whether we can satisfy all the constraints is a special case of the above decision problem since we can set $k = m$, but in some cases we consider this simpler decision version, particularly when we want to show hardness.

In the case of OR constraints, the CSP and MAX-CSP problems correspond to the more widely known CNF-SAT and MAX-CNF-SAT problems. In this case we call the constraints *clauses*. When the constraint function is AND, the MAX-CSP problem is called MAX-DNF-SAT. In that case, the constraints are called *terms*. The problem MAX-PARITY is also known as MAX-LIN-2 (satisfy a maximum number of given linear equations modulo 2).

For a CSP ψ, the incidence graph G^*_ψ is defined as the bipartite graph where we construct one vertex v_i for each (unsigned) variable x_i and one vertex u_j for each constraint C_j and connect v_i with u_j if $x_i \in C_j$.

We are interested in parameterizations of the incidence graph $\mathrm{p}(\mathrm{G}^*_\psi)$ (or simply p^* if G^*_ψ is clear from the context), where p is a structural parameter of G^*_ψ. We are mostly interested in the two most widely studied graph parameters, treewidth tw^* and clique-width cw^*. We refer the reader to standard parameterized complexity textbooks for the definitions, as well as the definitions of standard parameterized complexity terminology used in this paper [5].

3 CNF-SAT and MAX-CNF-SAT

In this section, we consider one of the most fundamental problems in computer science: the satisfiability problem for CNF formulas, which can be viewed as a constraint satisfaction problem where the only allowed constraints are clauses, that is, ORs of literals. An optimal solution for MAX-CNF-SAT can be computed in FPT when parameterized by the treewidth tw^* of the incidence graph [1], and hence CNF-SAT can be solved in the same time. When parameterized by the clique-width cw^* of the incidence graph, all known exact

algorithms for CNF-SAT and MAX-CNF-SAT run in XP time [22, 19]. Moreover, we don't expect these problems to be in FPT since they are both W[1]-hard parameterized by cw* [17].

In Section 3.1, we construct an approximation scheme for MAX-CNF-SAT that runs in FPT time. Intuitively, our algorithm works as follows: given a formula ϕ with 'small' incidence clique-width, we first examine the formula to see if it contains many or few 'large' clauses. If the formula contains relatively few large clauses, then we simply disregard them. We then know that the incidence graph does not contain 'large' bi-cliques, so by a theorem of Gurski and Wanke [10] the remaining formula has small treewidth and we can solve the problem. If on the other hand the original formula contains a large number of large clauses, then we observe that we can rely on a random assignment to satisfy almost everything.

In Section 3.2, we explore a class of CSP instances that is smaller than the class of bounded incidence clique-width instances; our goal is to understand which incidence graph parameter is responsible for the transition from FPT to W[1]. To this end, we have to look for a graph parameter that is bounded by a function of cw* (where it's hard) but can leave the tw* unbounded (where it's FPT). In fact, [17] shows that the problem is W[1]-hard parameterized by the modular treewidth mtw* of the incidence graph, which lies between cw* and tw*.[3] We study the incidence *neighborhood diversity* nd*, which is incomparable to tw*; however, mtw* is bounded when nd* is. We prove that CNF-SAT remains W[1]-hard parameterized by nd*.

▶ **Definition 1.** A graph $G(V, E)$ has neighborhood diversity $\mathrm{nd}(G) = k$ if we can partition V into k sets $V_1, \ldots, V_k$ such that, for all $v \in V$ and all $i \in \{1, \ldots, k\}$, either v is adjacent to every vertex in V_i or it is adjacent to none of them.

In other words, $\mathrm{nd}(G) = k$ if V can be partitioned into k modules that are either cliques or independent sets.

Formulas whose incidence graph has neighborhood diversity at most k seem very restrictive: there are only at most k variable- and clause-types, where all variables of the same type belong to the same clauses and all clauses of the same type involve the same variables. This class of formulas is a subset of formulas with $\mathrm{mtw}^* \leq k$ because contracting all modules leaves a graph of *order* at most k, which trivially has treewidth at most k.

3.1 Approximation Algorithm parameterized by clique-width

▶ **Theorem 2.** *There is a randomized algorithm that, given a CNF formula ψ with n variables, m clauses, and incidence clique-width* cw*, runs in time $f(\epsilon, \mathrm{cw}) \cdot \mathrm{poly}(n+m)$, and outputs a truth assignment that satisfies at least $(1-\epsilon) \cdot \mathrm{OPT}$ clauses in expectation.*

We formulate the following basic lemma about probability distributions.

▶ **Lemma 3.** *For all $\epsilon, L > 0$ there is a $c = c(\epsilon, L) > 0$ such that all $c' \geq c$ and all sequences $p_1, \ldots, p_{c'} \geq 0$ with $\sum_{i=1}^{c'} p_i \leq 1$ have an index $d \leq c/L$ with the property*

$$p_{[d, L\cdot d]} \doteq \sum_{j=d}^{L\cdot d} p_j < \epsilon\,.$$

[3] A graph of *bounded modular treewidth* is a graph of bounded treewidth after merging modules into a single vertex, where a module is a set of vertices with same neighborhood outside of the set. In fact, the reduction in [17] constructs a formula whose incidence graph has small feedback vertex set after contracting modules.

Proof. Let $\epsilon, L > 0$. We set $c = c(\epsilon, L)$ below. Assume for contradiction that $p_{[d,L\cdot d]} \geq \epsilon$ holds for all $d \in [1, c/L]$. If there are $1/\epsilon + 1$ disjoint intervals $[a_1, L \cdot a_1], \ldots, [a_{1/\epsilon+1}, L \cdot a_{1/\epsilon+1}] \subseteq [1, c]$, then we arrive at a contradiction with the fact that the p_i's are non-negative and sum to at most one. Clearly there exists a constant $c = c(\epsilon, L)$ such that $1/\epsilon + 1$ disjoint intervals of the form $[a, La]$ fit into $[1, c]$. This proves the claim. ◀

For an arbitrary given $\epsilon > 0$, we fix $L = \epsilon^{-4}$. We use Lemma 3 as follows: For a CNF formula ψ, we define p_i as the fraction of clauses of size i, that is,

$$p_i \doteq \frac{\left|\left\{ C \in \psi \;\middle|\; |C| = i \right\}\right|}{|\psi|}.$$

Then Lemma 3 gives us a number $d \leq c(\epsilon)$ such that the total fraction of clauses whose size is between d and $\epsilon^{-4}d$ is bounded by ϵ. It is now natural to partition all clauses into short, medium, and long clauses. More precisely, we define $\psi = \psi^{<d} \dot\cup \psi^{[d,D]} \dot\cup \psi^{>D}$ for $D = \epsilon^{-4}d$ as follows:

$$\begin{aligned}
\psi^{<d} &\doteq \left\{ C \in \psi \;\middle|\; |C| < d \right\}, \\
\psi^{[d,D]} &\doteq \left\{ C \in \psi \;\middle|\; d \leq |C| \leq D \right\}, \text{and} \\
\psi^{>D} &\doteq \left\{ C \in \psi \;\middle|\; |C| > D \right\}.
\end{aligned}$$

An immediate corollary to Lemma 3 is thus that we can choose $d \leq c(\epsilon)$ in such a way that $|\psi^{[d,D]}| \leq \epsilon|\psi|$.

▶ **Corollary 4.** *For all $\epsilon > 0$ there is some $c = c(\epsilon) > 0$ such that all CNF formulas ψ have some $d = d(\epsilon) \in [1, c]$ with $|\psi^{[d,\epsilon^{-4}d]}| \leq \epsilon \cdot |\psi|$.*

If $\psi^{[d,D]} = \emptyset$ holds for $D = \epsilon^{-4}d$ and $d \in [1, c(\epsilon)]$, we say that ψ is *ϵ-well separated*. We call ψ *ϵ'-balanced* if, in addition, we have $|\psi^{<d}| \geq \epsilon' m$ and $|\psi^{>D}| \geq \epsilon' m$.

▶ **Lemma 5.** *Let ψ be an ϵ-well separated formula (and thus $V = V(\psi^{<d}) \cup V(\psi^{>D})$).*

Then, for each subset $\hat\psi \subseteq \psi^{>D}$ with $|\hat\psi| > \epsilon^2 m$, there is a variable y such that $|\psi_y^{<d}| \leq \epsilon^2 |\hat\psi_y|$.

That is, for every set $\hat\psi$ that contains a significant fraction of long clauses, there is a variable that occurs $|\hat\psi_y|$ times in $\hat\psi$, but only at most an ϵ^2-fraction of that in the short clauses.

Proof. Let $\hat\psi \subseteq \psi^{>D}$ with $|\hat\psi| > \epsilon^2 m$. Note that the total number of literal occurrences in $\hat\psi$ is $\operatorname{occ}(\hat\psi) > D \cdot \epsilon^2 \cdot m = \epsilon^{-2} dm$. In contrast, $\operatorname{occ}(\psi^{<d}) < dm$. Now suppose that there was no variable y with the claimed properties, that is, suppose that every variable y satisfies $|\psi_y^{<d}| > \epsilon^2 |\hat\psi_y|$. Then the total number of variable occurrences in $\psi^{<d}$ can be bounded from below as follows:

$$\operatorname{occ}(\psi^{<d}) = \sum_y |\psi_y^{<d}| > \sum_y \epsilon^2 |\hat\psi_y| = \epsilon^2 \operatorname{occ}(\hat\psi) > d \cdot m.$$

This yields a contradiction and thus proves the claim. ◀

Proof of Theorem 2. The algorithm A works as follows. Let $\epsilon' = \epsilon^2$, and we assume w.l.o.g. that $\epsilon < 1/8$. Given a CNF formula ϕ, we compute an ϵ'-well separated formula ψ by dropping all clauses in $\phi^{[d,D]}$. Corollary 4 guarantees that the fraction of deleted clauses is

bounded by ϵ'. If ψ is not $\epsilon/2$-balanced, we discard the smaller side (with fewer clauses) and only handle the larger one: If $\psi^{<d}$ is the larger side, we compute an optimal assignment for $\psi^{<d}$ in FPT time, by using the result of Gurski and Wanke [10]. This way the total number of unsatisfied clauses is at most $\epsilon m/2$, and together with the unsatisfied clauses due to applying Corollary 4, the total number of unsatisfied clauses is smaller than ϵm. Since OPT $> m/2$, we get the approximation guarantee.

If $\psi^{>D}$ is the larger side, we use a random assignment. This way, at most $\epsilon m/2$ clauses from $\psi^{<d}$ are violated, and in expectation at most a 2^{-D} fraction of clauses from $\psi^{>D}$ are violated. Since 2^{-D} is smaller than $\epsilon/4$, we conclude that – together with unsatisfied clauses due to applying Corollary 4 – at least $(1-\epsilon)m$ clauses are satisfied in expectation.

This finishes the analysis of unbalanced formulas, and in the remaining proof we may assume that ψ is $\epsilon/2$-balanced. To handle this case, we determine a set of variables Y such that

- there are at most $\epsilon m/4$ short clauses with variables from Y and
- there are at most $\epsilon^2 m$ long clauses that contain $\leq 1/\epsilon$ variables from Y.

Before we construct Y, let us verify that the properties of Y imply the correctness of the theorem. Our algorithm computes a satisfying assignment of the short clauses without variables from Y, again using the result of Gurski and Wanke [10]. Afterwards it assigns values uniformly at random to the variables in Y.

There are at most $\epsilon' m = \epsilon^2 m$ unsatisfied clauses due to applying Corollary 4, $\epsilon m/4$ short clauses clauses that we did not consider when satisfying clauses from $\psi^{<d}$, and $\epsilon^2 m$ clauses from $\psi^{>D}$ that we did not consider in the random assignment. Additionally, in expectation there are less than $2^{-1/\epsilon} m$ clauses left unsatisfied from the remaining $|\psi^{>D}| - \epsilon^2 m$ clauses from $\psi^{>D}$. Since we assumed that $\epsilon < 1/8$, the theorem follows.

To construct the set Y, we iteratively apply Lemma 5 with the parameter $\epsilon/4$. Initially, we set $\hat{\psi} = \psi^{>D}$. In each iteration, we identify a variable y according to the lemma and add the variable to Y. In the subsequent iterations, we mark y to be inactive and handle it as if it was not contained in any clause. Whenever we identify a clause C that has at least $1/\epsilon$ inactive variables (i. e., variables from Y), we remove C from $\hat{\psi}$. We continue this process until $|\hat{\psi}| \leq \epsilon^2$. Note that applying Lemma 5 for $\epsilon/4$ but having an ϵ'-well separated formula ensures that at all times, all clauses in $\hat{\psi}$ have sufficiently many literals to apply Lemma 5. Therefore the process terminates and the generated set Y has the aimed-for properties since $|Y| \leq m/\epsilon$. ◀

3.2 Hardness parameterized by neighborhood diversity

A *constraint* on r variables is a relation $R \subseteq \{0,1\}^r$. We define the unary constraints $U_0 = \{0\}$ and $U_1 = \{1\}$, which corresponds to clauses $(\neg x)$ and (x), respectively. We define the equality $=$ and disequality $\neq$ constraints on two groups of Boolean variables $x = x_1 \dots x_n$ and $y = y_1 \dots y_n$ in infix notation in the usual way: For an assignment a to the x- and y-variables, we say that $a \models x = y$ if and only if, for all $i \in [n]$, we have $a(x_i) = a(y_i)$, that is, the assignment sets x_i to the same value as y_i; as usual, $x \neq y$ is interpreted as the negation of $x = y$.

▶ **Lemma 6.** *CNF-SAT parameterized by* nd* *is* W[1]*-hard, where* nd* *is the neighborhood diversity of the input's incidence graph.*

Proof. We devise an FPT-reduction from k-Multicolored Clique to CNF-SAT. Given a k-partite graph G, whose parts $V_1, \dots, V_k$ all have the same size n, we construct k groups of

variables $x_1, \ldots, x_k$, which together are supposed to represent a k-clique in G, should one exist. Each group x_i consists of $\log n$ Boolean variables and represents the supposed clique's vertex in the part V_i. Without loss of generality, we assume that $\log n$ is an integer.

Starting from the empty CNF formula, we construct a formula ϕ on the x-variables as follows. First choose, for each $i \in [k]$, an arbitrary bijection $\mathrm{bin}_i : V_i \to \{0,1\}^{\log n}$ that maps any vertex $u \in V_i$ to its binary representation $\mathrm{bin}(u)$; for convenience, we drop the index i. For each $i, j \in [k]$ with $i < j$, and for each non-edge $(u, v) \notin E(V_i, V_j)$ between V_i and V_j, we add the following constraint $C_{i,j,u,v}$ to ϕ:

$$x_i x_j \neq \mathrm{bin}(u)\mathrm{bin}(v)\,.$$

Clearly, this constraint excludes exactly one of the $2^{2\log n}$ possible assignments to $x_i x_j$, and so it can be written as an OR of literals of the x-variables. In the end, ϕ is a CNF formula with $|E(G)|$ clauses.

For the completeness of the reduction, let $v_i \in V_i$ for all $i \in [k]$ be such that $v_1, \ldots, v_k$ is a clique in G. For all $i \in [k]$, set $x_i = \mathrm{bin}(v_i)$. This assignment satisfies all constraints: for all $(u, v) \notin E(V_i, V_j)$, we have that $\mathrm{bin}(v_i)\mathrm{bin}(v_j) \neq \mathrm{bin}(u)\mathrm{bin}(v)$ because (v_i, v_j) is an edge and (u, v) is not, and bin is a bijection.

For the soundness of the reduction, let $a_1, \ldots, a_k \in \{0,1\}^{k \log n}$ be a satisfying assignment of ϕ. For each $i \in [k]$, let v_i be the unique vertex in V_i for which $\mathrm{bin}(v_i) = a_i$. Let $i, j \in [k]$ with $i < j$. Since the assignment satisfies all constraints of ϕ, it must be the case that (v_i, v_j) is an edge in G. For if it was a non-edge, its corresponding constraint in ϕ would have excluded the assignment $a_i a_j$ for $x_i x_j$. Hence $v_1, \ldots, v_k$ is a clique in G.

It remains to argue that the neighborhood diversity of the incidence graph of ϕ is at most $k + \binom{k}{2}$. The modules of the incidence graph are the variable group x_h for each $h \in [k]$ and the clause group $\{C_{i,j,u,v}\}$ for each $i, j \in [k]$ with $i < j$. Indeed, the incidence graph between x_h and $C_{i,j,*,*}$ is a bipartite clique if $h \in \{i, j\}$, and otherwise it is an independent set.

We constructed an FPT-reduction from the W[1]-complete problem Multicolored Clique to CNF-SAT parameterized by nd^*, which finishes the proof of the theorem. ◀

4 From Treewidth to Clique-width

In the previous section, we have seen that MAX-CNF-SAT is fixed-parameter tractable when parameterized by tw^*, which is a sparse graph parameter, and it is W[1]-hard to compute exactly and has an FPT-AS when parameterized by nd^*, which is a dense graph parameter. In this section we observe that the transition from sparse to dense parameters has different effects on the complexity of MAX-CSP, depending on which types of constraints are allowed.

By modifying our reduction for CNF-SAT we show that MAX-DNF-SAT, the problem of maximizing the number of satisfied AND constraints is W[1]-hard parameterized nd^*. Furthermore, because the maximum number of constraints that could be satisfied in our reduction is also bounded by some function of the parameter, we show that the problem does not have an FPT-AS unless FPT=W[1]. Thus, while MAX-DNF-SAT is FPT parameterized by tw^*, it does not even appear to have an FPT approximation scheme when parameterized by nd^*.

▶ **Theorem 7.** *Suppose that there exists an* FPT-AS *which, given $\epsilon > 0$ and an instance ϕ of* MAX-DNF-SAT*, computes a $(1-\epsilon)$-approximate solution and runs in time $f(\mathrm{nd}^*, \epsilon) \cdot \mathrm{poly}(n)$, where nd^* is the neighborhood diversity of the incidence graph of ϕ. Then* FPT = W[1].

The proof is similar that of Theorem 6 and deferred to the full version.

When parameterized by tw*, MAX-CNF-SAT and MAX-DNF-SAT are both FPT, and when parameterized by a dense graph parameter, such as nd*, the former problem is hard but approximable while the latter problem is hard even to approximate. We next consider natural constraint types where the corresponding CSPs stay FPT both for sparse as well as dense incidence graph parameters. MAX-PARITY wants to find an assignment that satisfies the maximum number of linear equations modulo two. While deciding whether there is an assignment that satisfies all equations is in P (by Gauss elimination), the maximization version is a typical APX-hard problem [11]. Here we show that computing the optimal solution of MAX-PARITY is FPT, regardless of whether the parameter is the treewidth or the clique-width of the incidence graph. Our intuition for why MAX-PARITY appears to be so much easier than CNF-SAT is that negations are (almost) irrelevant, and so the incidence graph seems to capture most of the structure relevant to the complexity of the CSP instance.

▶ **Theorem 8.** *Given an instance ϕ for* MAX-PARITY*, we can find an optimal solution in time $f(\text{cw}^*)|\phi|^{O(1)}$, where* cw* *is the clique-width of the incidence graph of ϕ.*

The proof relies on the meta-theorem of [3] and appears in the full version.

5 Majority and Threshold CSPs

In this section we describe our results for CSPs where each constraint is a MAJORITY constraint. Here a constraint MAJORITY($\{\ell_1, \ldots, \ell_d\}$) stipulates that at least $d/2$ of the literals ℓ_i are set to true. We denote the resulting CSP problem with MAJORITY, and the resulting MAX-CSPwith MAX-MAJORITY.

5.1 Hardness of exact algorithms

We parameterize MAJORITY by the size fvs* of the smallest feedback vertex set, or by the neighborhood diversity nd* of the instance's incidence graph. These parameterized problems turn out to be hard, even for the special case of MAJORITY constraints. Thus, neither dense nor sparse incidence graph parameters appear to put the problem in FPT.

▶ **Theorem 9.** MAJORITY *parameterized by the incidence feedback vertex set number* fvs* *is* W[1]*-hard.*

▶ **Theorem 10.** MAJORITY *parameterized by the incidence neighborhood diversity* nd* *is* W[1]*-hard.*

While the former theorem is proved by a technical reduction from Multicolored Clique, the latter is established using a straightforward reduction from Lemma 6. Proofs appear in the full version.

5.2 Exact Algorithm parameterized by vertex cover

Motivated by the negative result of Theorem 9 we now investigate the complexity of MAJORITY for more restricted parameters. The first parameter we consider is the vertex cover of the incidence graph. This is a natural, though quite restrictive, parameter which is often considered for problems which are W-hard for treewidth.

▶ **Theorem 11.** MAX-THRESHOLD *parameterized by the incidence vertex cover* vc* *is* FPT.

The proof appears in the full version, and it works by reducing the problem to integer linear programming parameterized by the number of variables, which is FPT.

5.3 Approximation Algorithm parameterized by feedback vertex set

The results of Theorem 9 naturally pose the following question: can we evade the W-hardness of MAJORITY by designing an FPT-AS for the problem? In this section, though we do not resolve this question, we give some first positive indication that this may be possible. We consider THRESHOLD parameterized by the incidence graph's feedback vertex set (that is, the number of vertices that need to be deleted to make the graph acyclic). This is a natural, well-studied parameter that generalizes vertex cover but is a restriction of treewidth. It is also connected to the concept of back-door sets to acyclicity, which is well-studied in the parameterized CSP literature [16, 7].

Observe that approximating this CSP is non-trivial, since MAX-MAJORITY with constraints of arity two already generalizes Max-2SAT, and is hence APX-hard. On the other hand, MAX-MAJORITY can easily be 2-approximated by considering any assignment and its negation. Hence, the natural goal here is an approximation ratio of $(1-\epsilon)$. Using Corollary 11 as a sub-routine we achieve this with an FPT-AS.

▶ **Theorem 12.** *There exists an* FPT-AS *which, given* $\epsilon > 0$ *and an instance* ϕ *of* MAX-THRESHOLD*, computes a* $(1-\epsilon)$*-approximate solution and runs in time* $f(\text{fvs}^*, \epsilon) \cdot \text{poly}(n)$*, where* fvs^* *is the size of the smallest feedback vertex set of the incidence graph of* ϕ*.*

References

1 Michael Alekhnovich and Alexander A. Razborov. Satisfiability, branch-width and Tseitin tautologies. *Computational Complexity*, 20(4):649–678, 2011.

2 Per Austrin and Subhash Khot. A characterization of approximation resistance for even k-partite csps. In Robert D. Kleinberg, editor, *Innovations in Theoretical Computer Science, ITCS'13, Berkeley, CA, USA, January 9-12, 2013*, pages 187–196. ACM, 2013.

3 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.

4 Nadia Creignou. A dichotomy theorem for maximum generalized satisfiability problems. *Journal of Computer and System Sciences*, 51(3):511–522, 1995.

5 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

6 Serge Gaspers and Stefan Szeider. Kernels for global constraints. In Toby Walsh, editor, *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 540–545. IJCAI/AAAI, 2011.

7 Serge Gaspers and Stefan Szeider. Backdoors to acyclic SAT. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming – 39th International Colloquium, ICALP 2012, Warwick, UK, July 9–13, 2012, Proceedings, Part I*, volume 7391 of *Lecture Notes in Computer Science*, pages 363–374. Springer, 2012.

8 Serge Gaspers and Stefan Szeider. Guarantees and limits of preprocessing in constraint satisfaction and reasoning. *Artif. Intell.*, 216:1–19, 2014.

9 Martin Grohe. The structure of tractable constraint satisfaction problems. In Rastislav Kralovic and Pawel Urzyczyn, editors, *MFCS 2006, Stará Lesná, Slovakia, August 28-September 1, 2006, Proceedings*, volume 4162 of *Lecture Notes in Computer Science*, pages 58–72. Springer, 2006.

10 Frank Gurski and Egon Wanke. The tree-width of clique-width bounded graphs without $K_{n,\,n}$. In Ulrik Brandes and Dorothea Wagner, editors, *Graph-Theoretic Concepts in*

Computer Science, 26th International Workshop, WG 2000, Konstanz, Germany, June 15-17, 2000, Proceedings, volume 1928 of *Lecture Notes in Computer Science*, pages 196–205. Springer, 2000.

11 Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001.

12 Sanjeev Khanna, Madhu Sudan, and David P. Williamson. A complete classification of the approximability of maximization problems derived from boolean constraint satisfaction. In Frank Thomson Leighton and Peter W. Shor, editors, *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 11–20. ACM, 1997.

13 Subhash Khot and Rishi Saket. Approximating csps using LP relaxation. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming – 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6–10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 822–833. Springer, 2015.

14 Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, 2012.

15 Dániel Marx. Parameterized complexity and approximation algorithms. *Comput. J.*, 51(1):60–78, 2008.

16 Sebastian Ordyniak, Daniël Paulusma, and Stefan Szeider. Satisfiability of acyclic and almost acyclic CNF formulas. *Theor. Comput. Sci.*, 481:85–99, 2013.

17 Daniël Paulusma, Friedrich Slivovsky, and Stefan Szeider. Model counting for CNF formulas of bounded modular treewidth. In Natacha Portier and Thomas Wilke, editors, *STACS 2013, February 27 – March 2, 2013, Kiel, Germany*, volume 20 of *LIPIcs*, pages 55–66. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013.

18 Reinhard Pichler, Stefan Rümmele, Stefan Szeider, and Stefan Woltran. Tractable answer-set programming with weight constraints: bounded treewidth is not enough. *TPLP*, 14(2):141–164, 2014.

19 Sigve Hortemo Sæther, Jan Arne Telle, and Martin Vatshelle. Solving maxsat and #sat on structured CNF formulas. In Carsten Sinz and Uwe Egly, editors, *SAT 2014 – Vienna, Austria, July 14–17, 2014. Proceedings*, volume 8561 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2014.

20 Marko Samer and Stefan Szeider. Constraint satisfaction with bounded treewidth revisited. *J. Comput. Syst. Sci.*, 76(2):103–114, 2010.

21 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.

22 Friedrich Slivovsky and Stefan Szeider. Model counting for formulas of bounded clique-width. In Leizhen Cai, Siu-Wing Cheng, and Tak Wah Lam, editors, *ISAAC 2013, Hong Kong, China, Proceedings*, volume 8283 of *Lecture Notes in Computer Science*, pages 677–687. Springer, 2013.

23 Stefan Szeider. On fixed-parameter tractable parameterizations of SAT. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 188–202. Springer, 2003.

24 Stefan Szeider. Not so easy problems for tree decomposable graphs. *CoRR*, abs/1107.1177, 2011.

25 Stefan Szeider. The parameterized complexity of constraint satisfaction and reasoning. In Hans Tompits, Salvador Abreu, Johannes Oetsch, Jörg Pührer, Dietmar Seipel, Masanobu

Umeda, and Armin Wolf, editors, *INAP 2011, and WLP 2011, Vienna, Austria, September 28-30, 2011, Revised Selected Papers*, volume 7773 of *Lecture Notes in Computer Science*, pages 27–37. Springer, 2011.

26 Stefan Szeider. The parameterized complexity of k-flip local search for SAT and MAX SAT. *Discrete Optimization*, 8(1):139–145, 2011.

27 Luca Trevisan. Inapproximability of combinatorial optimization problems. *Electronic Colloquium on Computational Complexity (ECCC)*, 2004.

Enumerating Minimal Connected Dominating Sets in Graphs of Bounded Chordality*

Petr A. Golovach[1], Pinar Heggernes[1], and Dieter Kratsch[2]

1 **Department of Informatics, University of Bergen, Norway**
{petr.golovach,pinar.heggernes}@ii.uib.no
2 **Université de Lorraine, LITA, Metz, France**
dieter.kratsch@univ-lorraine.fr

Abstract

Listing, generating or enumerating objects of specified type is one of the principal tasks in algorithmics. In graph algorithms one often enumerates vertex subsets satisfying a certain property. We study the enumeration of all minimal connected dominating sets of an input graph from various graph classes of bounded chordality. We establish enumeration algorithms as well as lower and upper bounds for the maximum number of minimal connected dominating sets in such graphs. In particular, we present algorithms to enumerate all minimal connected dominating sets of chordal graphs in time $O(1.7159^n)$, of split graphs in time $O(1.3803^n)$, and of AT-free, strongly chordal, and distance-hereditary graphs in time $O^*(3^{n/3})$, where n is the number of vertices of the input graph. Our algorithms imply corresponding upper bounds for the number of minimal connected dominating sets for these graph classes.

1998 ACM Subject Classification G.2.2 Graph Theory, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Minimal connected dominating set, exact algorithms, enumeration

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.307

1 Introduction

Listing, generating or enumerating objects of specified type and properties has important applications in various domains of computer science, such as data mining, machine learning, and artificial intelligence, as well as in other sciences, especially biology. In particular, enumeration algorithms whose running time is measured in the size of the input have gained increasing interest recently. The reason for this is two-fold. Firstly, many exact exponential-time algorithms for the solution of NP-hard problems rely on such enumeration algorithms. Sometimes the fastest known algorithm to solve an optimization problem is by simply enumerating all minimal or maximal feasible solutions (e.g., for subset feedback vertex sets [15]), whereas other times the enumeration of some objects is useful for algorithms for completely different problems (e.g., enumeration of maximal independent sets in triangle-free graphs for computing graph homomorphisms [14]). Secondly, the running times of such enumeration algorithms very often imply an upper bound on the maximum number of enumerated objects a graph can have. This is a field of research that has long history within combinatorics, and enumeration algorithms provide an alternative way to prove such

* The research leading to these results has received funding from the Research Council of Norway and the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 267959.

10th International Symposium on Parameterized and Exact Computation (IPEC 2015).
Editors: Thore Husfeldt and Iyad Kanj; pp. 307–318

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

combinatorial bounds. In fact, several classical examples exist in this direction, of which one of the most famous is perhaps that of Moon and Moser [25] who showed that the maximum number of maximal independent sets in a graph on n vertices is $3^{n/3}$. Although the arguments of [25] were purely combinatorial, the same bound is also achieved by an enumeration algorithm with running time $O^*(3^{n/3})$, where the O^*-notation suppresses polynomial factors.

The mentioned result on the number of maximal independent sets is tight, as there is a graph that has exactly $3^{n/3}$ maximal independent sets, namely a disjoint union of $n/3$ triangles. However, for many upper bounds, no such matching lower bound is known, and hence for the maximum number of many objects there is a gap between the known upper and lower bounds. This motivates the study of enumeration of objects in graphs belonging to various graph classes. For example, the maximum number of minimal dominating sets in graphs is known to be at most 1.7159^n [13], however no graph having more than 1.5704^n minimal dominating sets is known. On the other hand, on many graph classes matching upper and lower bounds can be shown on the maximum number of minimal dominating sets [7, 9]. Furthermore, even if the bound on general graphs is tight, a better bound might exist for graph classes, which might be useful algorithmically and interesting combinatorially. For example, the maximum number of maximal independent sets in triangle-free graphs is at most $2^{n/2}$ and they can be listed in time $O^*(2^{n/2})$ [5], which was used in the above mentioned algorithm for homomorphisms [14]. As a consequence, there has been extensive research in this direction recently, both on general graphs and in particular on graph classes. Examples of algorithms for the enumeration and combinatorial lower and upper bounds on graph classes exist for minimal feedback vertex sets, minimal subset feedback vertex sets, minimal dominating sets, minimal separators, and potential maximal cliques [7, 8, 9, 12, 15, 17, 19, 20, 21].

In this paper we initiate the study of the enumeration and maximum number of minimal *connected* dominating sets in a given graph. Interestingly, the best known upper bound for the maximum number of minimal connected dominating sets in an arbitrary n-vertex graph is 2^n, i.e., the trivial one. The best lower bound we achieve in this paper is $3^{(n-2)/3}$, and thus the gap between the known lower and upper bounds is huge on arbitrary graphs. Furthermore, although connected dominating sets have been subject to extensive study when it comes to optimization and decision variants, their enumeration has been left completely unattended. In fact computing a minimum connected dominating set is one of the classical NP-hard problems already mentioned in the monograph of Garey and Johnson [18]. The best known running time of an algorithm solving this problem is $O(1.8619^n)$ [1], which is surprisingly larger than the best known lower bound $3^{(n-2)/3} \approx 1.4423^n$.

The results that we present in this paper are summarized in the following table, where n is the number of vertices and m is the number of edges of an input graph belonging to the given class.

Graph Class	Lower Bound	Upper Bound	Enumeration Algorithm
chordal	$3^{(n-2)/3}$	$O(1.7159^n)$	$O(1.7159^n)$
split	1.3195^n[7]	1.3803^n	$O(1.3803^n)$
cobipartite	1.3195^n[7]	1.3803^n	$O(1.3803^n)$
interval	$3^{(n-2)/3}$	$3^{(n-2)/3}$	$O^*(3^{n/3})$
AT-free	$3^{(n-2)/3}$	$O^*(3^{(n-2)/3})$	$O^*(3^{n/3})$
strongly chordal	$3^{(n-2)/3}$	$3^{n/3}$	$O^*(3^{n/3})$
distance-hereditary	$3^{(n-2)/3}$	$3^{n/3} \cdot n$	$O^*(3^{n/3})$
cograph	m	m	$O(m)$

2 Preliminaries

We consider finite undirected graphs without loops or multiple edges. For each of the graph problems considered in this paper, we let $n = |V(G)|$ and $m = |E(G)|$ denote the number of vertices and edges, respectively, of the input graph G. For a graph G and a subset $U \subseteq V(G)$ of vertices, we write $G[U]$ to denote the subgraph of G induced by U. We write $G - U$ to denote the subgraph of G induced by $V(G) \setminus U$, and $G - u$ if $U = \{u\}$. A set $U \subseteq V(G)$ is *connected* if $G[U]$ is a connected graph. For a vertex v, we denote by $N_G(v)$ the *(open) neighborhood* of v, i.e., the set of vertices that are adjacent to v in G. The *closed neighborhood* $N_G[v] = N_G(v) \cup \{v\}$. For a set of vertices $U \subseteq V(G)$, $N_G[U] = \cup_{v \in U} N_G[v]$ and $N_G(U) = N_G[U] \setminus U$. We say that P is a (u, v)-*path* if P is a path that joins u and v. The vertices of P different from u and v are the *inner* vertices of P. The *distance* $\mathrm{dist}_G(u, v)$ between vertices u and v of G is the number of edges on a shortest (u, v)-path. A path (cycle) P is *induced* if it has no *chord*, i.e., there is no edge of G incident to any two vertices of P that are not adjacent in P. The *chordality* $\mathrm{chord}(G)$ of a graph G is the length of a longest induced cycle in G; if G has no cycles, then $\mathrm{chord}(G) = 0$. A vertex v is a *cut vertex* of a connected graph G with at least two vertices if $G - v$ is disconnected.

For a non-negative integer k, a graph G is k-*chordal* if $\mathrm{chord}(G) \leq k$. A graph is *chordal* if it is 3-chordal. A graph G is *strongly chordal* if G is chordal and every cycle C of even length at least 6 in G has an *odd chord*, i.e., an edge that connects two vertices of C that are an odd distance apart from each other in the cycle. A graph G is *distance-hereditary* if for any connected induced subgraph H of G, $\mathrm{dist}_H(u, v) = \mathrm{dist}_G(u, v)$ for $u, v \in V(H)$. Distance hereditary graphs are 4-chordal. An *asteroidal triple (AT)* is a set of three pairwise non-adjacent vertices such that between each pair of them there is a path that does not contain a neighbor of the third. A graph is *AT-free* if it contains no AT. Consequently, AT-free graphs are 5-chordal.

A graph is a *split* graph if its vertex set can be partitioned in an independent set and a clique. Split graphs are chordal. A graph G is an *interval graph* if it is the intersection graph of a set of closed intervals on the real line, i.e., the vertices of G correspond to the intervals and two vertices are adjacent in G if and only if their intervals have at least one point in common. Interval graphs are strongly chordal and AT-free. A graph is a *cobipartite* graph if its vertex set can be partitioned into two cliques. A graph is a *cograph* if it has no induced path on 4 vertices. Cobipartite graphs and cographs are AT-free as well as 4-chordal.

Each of the above-mentioned graph classes can be recognized in polynomial (in most cases linear) time, and they are closed under taking induced subgraphs [4, 22]. See the monographs by Brandstädt et al. [4] and Golumbic [22] for more properties and characterizations of these classes and their inclusion relationships.

A vertex v of a graph G *dominates* a vertex u if $u \in N_G[v]$; similarly v dominates a set of vertices U if $U \subseteq N_G[v]$. For two sets $D, U \subseteq V$, D dominates U if $U \subseteq N_G[D]$. A set of vertices D is a *dominating set* of G if D dominates $V(G)$. A set of vertices D is a *connected dominating set* if D is connected and D dominates $V(G)$. A (connected) dominating set is *minimal* if no proper subset of it is a (connected) dominating set. Let $D \subseteq V(G)$, and let $v \in D$. Vertex u is a *private vertex*, or simply *private*, for vertex v (with respect to D) if u is dominated by v but is not dominated by $D \setminus \{v\}$. Clearly, a dominating set D is minimal if and only if each vertex of D has a private vertex. Notice also that a connected dominating set D is a minimal if and only if for any $v \in D$, v has a private vertex or $D \setminus \{v\}$ is disconnected, i.e., v is a cut vertex of $G[D]$. Observe that a vertex can be private for itself with respect to a dominating set D, but if D is a connected dominating set of size at least two, then any private u of v is a neighbor of v.

For technical reasons, we also consider *red-blue domination*. Let $\{R, B\}$ form a bipartition of the vertex set of a graph G. We refer to the vertices of R as the *red* vertices, the vertices of B as the *blue* vertices, and we say that G is a *red-blue graph*. A set of vertices $D \subseteq R$ is a *red dominating set* if D dominates B, and D is *minimal* if no proper subset of it dominates B. It is straightforward to see that $D \subseteq R$ is a minimal red dominating set if and only if D dominates B and each vertex of D has a private blue vertex.

We conclude this section by showing a lower bound for the maximum number of minimal connected dominating sets.

▶ **Proposition 1.** *There are interval and distance-hereditary graphs with at least* $3^{(n-2)/3}$ *minimal connected dominating sets.*

Proof. To obtain the bound for interval and distance-hereditary graphs, consider the graph G constructed as follows for a positive integer k.

- For $i \in \{1, \ldots, k\}$, construct a triple of pairwise adjacent vertices $T_i = \{x_i, y_i, z_i\}$.
- For $i \in \{2, \ldots, k\}$, join each vertex of T_{i-1} with every vertex of T_i by an edge.
- Construct two vertices u and v and edges ux_1, uy_1, uz_1 and vx_k, vy_k, vz_k.

Clearly, G has $n = 3k+2$ vertices. Notice that $D \subseteq V(G)$ is a minimal connected dominating set of G if and only if $u, v \notin D$ and $|D \cap T_i| = 1$ for $i \in \{1, \ldots, k\}$. Therefore, G has $3^k = 3^{(n-2)/3}$ minimal connected dominating sets. It remains to observe that G is both interval and distance-hereditary. ◀

3 Chordal graphs

In this section we shall heavily rely on minimal separators of graphs and minimal transversals of hypergraphs. A vertex set $S \subseteq V$ is a *separator* of the graph $G = (V, E)$ if $G - S$ is disconnected. A component C of $G - S$ is *full* if every vertex of S has a neighbor in C. A separator S of G is a *minimal separator* of G if $G - S$ has at least two full components. Minimal separators of graphs have been studied intensively in the last twenty years. They play a crucial role in minimal triangulations, and in solving problems like treewidth and minimum fill-in. For more information we refer to [23].

Let us start with a strong relationship between the minimal connected dominating sets of a graph and its minimal separators, established by Kante, Limouzy, Mary and Nourine [24]. First note that disconnected graphs have no connected dominating set, and all minimal connected dominating sets of a complete graph G are singletons $\{v\}$ with $v \in V(G)$. Now we define the *minsep hypergraph* $H = (V(H), E(H))$ of a graph $G = (V, E)$. The vertex set of H consists of all vertices of G belonging to some minimal separator of G, hence $V(H) \subseteq V(G)$. The hyperedges of H are exactly the minimal separators of G. Hence $|E(H)|$ is the number of minimal separators of G. Recall that a transversal of a hypergraph H is a vertex set $T \subseteq V(H)$ intersecting all hyperedges of H, i.e. $T \cap A \neq \emptyset$ for all $A \in E(H)$. Furthermore a transversal is minimal if no proper subset of it is a transversal.

▶ **Theorem 2** ([24])**.** *Let $G = (V, E)$ be a connected and non complete graph. Then $D \subseteq V$ is a minimal connected dominating set of G if and only if D is a minimal transversal of the minsep hypergraph H of G.*

To enumerate the minimal transversals of the minsep hypergraph of chordal graphs, we will be relying on a branching algorithm and its analysis which is due to Fomin, Grandoni, Pyatkin and Stepanov [13]. The main result of their paper is that a graph has at most 1.7159^n minimal dominating sets. The crucial result of the paper for us is the branching

algorithm to enumerate all minimal set covers of a set cover instance $(\mathcal{U}, \mathcal{S})$, where $\mathcal{U}$ is a universe and $\mathcal{S}$ is a collection of subsets of $\mathcal{U}$. When studying the algorithm and its analysis one observes that it can be applied to all set cover instances $(\mathcal{U}, \mathcal{S})$. Only at the very end of the analysis the obtained general bound is applied to the particular instances obtained from graphs satisfying $|\mathcal{U}| = |\mathcal{S}|$, which includes tailoring the weights to the case $|\mathcal{U}| = |\mathcal{S}|$. The interested reader may study Sections 3 and 4 of [13] to find the following implicit result.

▶ **Theorem 3** ([13]). *A set cover instance $(\mathcal{U}, \mathcal{S})$ has $O^*(\lambda^{|\mathcal{U}|+\alpha\cdot|\mathcal{S}|})$ minimal set covers, where $\lambda = 1.156154$ and $\alpha = 2.720886$. These minimal set covers can be enumerated in time $O^*(\lambda^{|\mathcal{U}|+\alpha|\mathcal{S}|})$.*

By Corollary 2 we are interested in enumerating the minimal transversals of a hypergraph $H = (V(H), E(H))$ with $V(H) = \{v_1, v_2, \dots v_s\}$ and $E(H) = \{E_1, E_2, \dots, E_t\}$ where $E_j \subseteq V(H)$ for all hyperedges E_j. It is well-known that enumerating the minimal transversals of a hypergraph H is equivalent to enumerating the minimal set covers of a set cover instance (corresponding to the dual hypergraph of H) constructed as follows. First we set $\mathcal{U} = E(H)$ and then $\mathcal{S}$ is a collection of sets $S(v_1), S(v_2), \dots S(v_s)$ such that for all $i \in \{1, 2, \dots s\}$ the set $S(v_i) \subseteq E(H)$ consists of all hyperedges E_j containing v_i. Consequently $|\mathcal{U}| = |E(H)|$ and $|\mathcal{S}| = |V(H)|$. By the construction enumerating the minimal set covers of the dual set cover instance $(\mathcal{U}, \mathcal{S})$ is equivalent to enumerating the minimal transversals of H. Consequently Theorem 3 implies

▶ **Corollary 4.** *A hypergraph $(V(H), E(H))$ has $O^*(\lambda^{|E(H)|+\alpha\cdot|V(H)|})$ minimal transversals, where $\lambda = 1.156154$ and $\alpha = 2.720886$. These minimal transversals can be enumerated in time $O^*(\lambda^{|E(H)|+\alpha|V(H)|})$.*

Now we are ready to consider the enumeration of minimal connected dominating sets on chordal graphs.

▶ **Theorem 5.** *A chordal graph has $O(1.7159^n)$ minimal connected dominating sets, and these sets can be enumerated in time $O(1.7159^n)$.*

Proof. Note that every chordal graph has a simplicial vertex, and no simplicial vertex belongs to a minimal separator. Furthermore a chordal graph has at most n minimal separators. Now let H be the minsep hypergraph of a chordal graph G. Let $s \geq 1$ be the number of maximal cliques containing a simplicial. Then $|V(H)| \leq |V(G)| - s = n - s$ and $|E(H)| \leq |V(G)| = n$. By Corollary 4, the number of minimal transversals of H is $O^*(\lambda^{|E(H)|+\alpha\cdot|V(H)|})$. Hence we can upper bound (up to a polynomial factor) the running time of the algorithm enumerating the minimal transversals and the number of minimal transversals by

$$\lambda^{|E(H)|+\alpha\cdot|V(H)|} \leq \lambda^{(1+\alpha)n} < 1.7159^n.$$

Consequently the number of minimal transversals of H is $O(1.7159^n)$, and they can be enumerated in time $O(1.7159^n)$. Finally by Corollary 2 these minimal transversals are precisely the minimal connected dominating sets of G. ◀

4 Split graphs and cobipartite graphs

We denote a split graph by $G = (C, I, E)$ to indicate that its vertex set $V(G)$ can be partitioned into a clique C and an independent set I. The following simple lemma will be crucial for our branching algorithm.

▶ **Lemma 6.** *Let $G = (C, I, E)$ be a split graph. Then D is a minimal connected dominating set of G if and only if either $D \subseteq C$ and D is minimal dominating set of G, or $|I| = 1$ and $D = I$ is a dominating set of G.*

Proof. Suppose $|I| \geq 2$. Then a minimal connected dominating set D of a split graph G cannot contain a vertex $v \in I$ since this would imply $w \in D$ for some $w \in C$ being adjacent to v. But then $D \setminus \{v\}$ would also be a connected dominating set, contradicting the minimality of D. ◀

Couturier et al. have shown that the maximum number of minimal dominating sets in a split graph is $3^{n/3}$, and that these sets can be enumerated in time $O^*(3^{n/3})$ [9]. Combined with Lemma 6, this implies that the same results hold for minimal connected dominating sets in split graphs. With a branching algorithm, given in the appendix, we are able to establish a significant improvement.

▶ **Theorem 7.** *A split graph has at most 1.3803^n minimal connected dominating sets, and these can be enumerated in time $O(1.3803^n)$.*

Theorem 7 implies an improvement on the number of minimal dominating sets for n-vertex cobipartite graphs. The previous best known bound is $O(1.4511^n)$ [9].

▶ **Corollary 8.** *A cobipartite graph has $O(1.3803^n)$ minimal (connected) dominating sets, and these sets can be enumerated in time $O(1.3803^n)$.*

Proof. Let $G = (C_1, C_2, E)$ be a cobipartite where its vertex set can be partitioned into cliques C_1 and C_2. Let D be a minimal dominating set of $G = (C_1, C_2, E)$. Then $D = \{x, y\}$ with $x \in C_1$ and $y \in C_2$, $D \subseteq C_1$ or $D \subseteq C_2$. Hence with the exception of the $O(n^2)$ minimal dominating sets $D = \{x, y\}$, all other minimal dominating sets are connected. There is a one-to-one relation of the minimal (connected) dominating sets of $G = (C_1, C_2, E)$ being a subset of C_1 and the minimal connected dominating sets of the split graph $G = (C_1, C_2, E)$ obtained by transforming C_2 into an independent set. Similarly, there is a one-to-one relation of the minimal (connected) dominating sets of $G = (C_1, C_2, E)$ being a subset of C_2 and the minimal connected dominating sets of the split graph $G = (C_2, C_1, E)$ obtained by transforming C_1 into an independent set. Hence the maximum number of minimal (connected) dominating sets of an n-vertex cobipartite graphs is equal to the maximum number of minimal connected dominating sets in an n-vertex split graph up to a polynomial factor. This together with Theorem 7 implies the corollary. ◀

The above one-to-one correspondence can also be used to obtain the best known lower bound for the maximum number of minimal connected dominating sets in an n-vertex split graph which is 1.3195^n, based on a lower bound construction for cobipartite graphs given in [7]. The following corollary will be useful in the next section.

▶ **Corollary 9.** *A red-blue graph G has $O(1.3803^n)$ minimal red dominating sets, and these can be enumerated in time $O(1.3803^n)$.*

Proof. Let $G = (R, B, E)$ be a red-blue graph. We construct a split graph $G' = (R, B, E)$ with clique R and independent set B. Then there is a one-to-one relation between the minimal red dominating sets of the red-blue graph G and the minimal connected dominating sets of the split graph G. Using this and Theorem 7 we get the result. ◀

5 AT-free graphs

We need the following folklore observation about the number of induced paths.

▶ **Lemma 10.** *For every pair of vertices u and v of a graph G, G has at most $3^{(n-2)/3}$ induced (u,v)-paths, and these paths can be enumerated in time $O^*(3^{n/3})$.*

Using Lemma 10, we can obtain the tight upper bound for the number of minimal connected dominating sets for interval graphs. Let G be an interval graph with at least two non-adjacent vertices. Consider an interval model of G, i.e., a collection of closed intervals on the real line corresponding to the vertices of G such that two vertices are adjacent in G if and only if their intervals intersect. Let u be the vertex of G corresponding to an interval with the leftmost right end-point and let v be the vertex corresponding to an interval with the rightmost left end-point. Notice that $u \neq v$ and u and v are not adjacent, because G is not a complete graph. It can be shown that $D \subseteq V(G)$ is a minimal connected dominating set of G if and only if D is the set of inner vertices of an induced (u,v)-path. This observation together with Lemma 10 immediately imply the following proposition.

▶ **Proposition 11.** *An interval graph has at most $3^{(n-2)/3}$ minimal connected dominating sets, and these sets can be enumerated in time $O^*(3^{n/3})$.*

Proposition 1 shows that the bound is tight. To extend it to AT-free graphs we need some additional terminology and auxiliary results. A path P in a graph G is a *dominating path* if $V(P)$ is a dominating set of G. A pair of vertices $\{u,v\}$ of G is a *dominating pair* if any (u,v)-path in G is a dominating path.

▶ **Lemma 12** ([6])**.** *Every connected AT-free graph has a dominating pair.*

We show the following properties of minimal connected dominating sets of AT-free graphs. Notice that if D is a connected dominating set of a graph G, then $G[D]$ is a connected AT-free graph and, therefore, $G[D]$ has a dominating pair by Lemma 12.

▶ **Lemma 13.** *Let D be a minimal connected dominating set of an AT-free graph G. Let $\{u,v\}$ be a dominating pair of $H = G[D]$ and suppose that $P = v_1 \dots v_k$, where $u = v_1$ and $v = v_k$, is a shortest (u,v)-path in H. Let $X_1 = N_G(\{v_1,v_2\}) \setminus N_G[v_4]$, $X_2 = N_G(\{v_{k-1},v_k\}) \setminus N_G[v_{k-3}]$, $Y_1 = N_G(X_1) \setminus N_G[\{v_1,\dots,v_4\}]$ and $Y_2 = N_G(X_2) \setminus N_G[\{v_{k-3},\dots,v_k\}]$. Then $D \subseteq N_G[V(P)]$ and if $k \geq 6$, the following holds:*

- **(i)** *$D \subseteq V(P) \cup X_1 \cup X_2$,*
- **(ii)** *for the (v_6,v_k)-subpath P_1 of P, $V(P_1) \cap N_G[\{v_1,\dots,v_4\} \cup Y_1] = \emptyset$, and for the (v_1,v_{k-5})-subpath P_2 of P, $V(P_2) \cap N_G[\{v_{k-3},\dots,v_k\} \cup Y_2]) = \emptyset$.*

Proof omitted for lack of space.

Now we are ready to enumerate the minimal connected dominating sets of AT-free graphs.

▶ **Theorem 14.** *An AT-free graph has $O^*(3^{(n-2)/3})$ minimal connected dominating sets, and these sets can be enumerated in time $O^*(3^{n/3})$.*

Proof. First, we show that there are at most $3^{n/3} \cdot n^9$ minimal connected dominating sets D such that $H = G[D]$ has a dominating pair $\{u,v\}$ with $\mathrm{dist}_G(u,v) \leq 8$ and enumerate these sets. Consider all the at most $\binom{n}{1} + \ldots + \binom{n}{9} \leq n^9$ possible choices of a pair of vertices $\{u,v\}$ and an induced path $P = v_1 \dots v_k$ with $u = v_1$ and $v = v_k$ such that $k \leq 9$. For each $\{u,v\}$ and P we enumerate the minimal connected dominating sets D such that $\{u,v\}$ is a dominating pair in $H = G[D]$ and P is a shortest (u,v)-path in H.

Let P be any induced path $P = v_1 \dots v_k$ with $u = v_1$ and $v = v_k$ such that $k \leq 9$. Consider the red-blue graph $G' = G - V(P)$, where the set of red vertices is $R = N_G(V(P))$ and the set of blue vertices is $B = V(G') \setminus R$. Let D be a minimal connected dominating set of G such that $\{u, v\}$ is a dominating pair of $H = G[D]$ and P is a shortest (u, v)-path in H. By Lemma 13, $D \subseteq N_G[V(P)]$. It is straightforward to see that $D \setminus V(P)$ is a red dominating set of G' that dominates all blue vertices and by minimality, $D \setminus V(P)$ is a minimal red dominating set. By Corollary 9, there are at most $1.3803^{|V(G')|} \leq 3^{n/3}$ such sets D and they can be enumerated in time $O(3^{n/3})$. We obtain that there are at most $3^{n/3} \cdot n^9$ minimal connected dominating sets D such that $H = G[D]$ has a dominating pair $\{u, v\}$ with $\mathrm{dist}_G(u, v) \leq 8$, and these sets can be enumerated in time $O(3^{n/3} \cdot n^9)$.

Now we enumerate minimal connected dominating sets D such that $H = G[D]$ has a dominating pair $\{u, v\}$ with $\mathrm{dist}_G(u, v) \geq 9$. Consider all the at most $\binom{n}{1} + \ldots + \binom{n}{10} \leq n^{10}$ possible choices of a pair of vertices $\{u, v\}$ and 2 disjoint induced paths $P_1 = x_1 \dots x_5$ and $P_2 = y_1 \dots y_5$ with $u = x_1$ and $v = y_5$. For each $\{u, v\}$, P_1 and P_2 we enumerate the minimal connected dominating sets D such that $\{u, v\}$ is a dominating pair in $H = G[D]$ and H has a shortest (u, v)-path $P = v_1 \dots v_k$ such that $v_i = x_i$ for $i \in \{1, \dots, 5\}$ and $v_i = y_{i+5-k}$ for $i \in \{k-4, \dots, k\}$.

Denote by $X_1 = N_G(\{x_1, x_2\}) \setminus N_G[x_4]$, $X_2 = N_G(\{y_4, y_5\}) \setminus N_G[y_2]$, $Y_1 = N_G(X_1) \setminus N_G[\{x_1, \dots, x_4\}]$ and $Y_2 = N_G(X_2) \setminus N_G[\{y_2, \dots, y_5\}]$. Consider the red-blue graph $G_1 = G[X_1 \cup X_2 \cup Y_1 \cup Y_2]$, where the set of red vertices $R = X_1 \cup X_2$ and the set of blue vertices $B = Y_1 \cup Y_2$. Let $n_1 = |V(G_1)|$. Let D be a minimal connected dominating set of G such that $\{u, v\}$ is a dominating pair of $H = G[D]$ and P is a shortest (u, v)-path in H. By Lemma 13, $D \subseteq N_G[V(P)]$ and $D' = D \setminus V(P) \subseteq X_1 \cup X_2$ is a red dominating set of G_1 that dominates all blue vertices and by minimality, D' is a minimal red dominating set. By Corollary 9, there are at most $1.3803^{n_1} \leq 3^{n_1/3}$ minimal red dominating sets in G_1, and they can be enumerated in time $O(3^{n_1/3})$.

Let $G_2 = G - (V(G_1) \cup \{x_1, \dots, x_4\} \cup \{y_2, \dots, y_5\})$ and let $n_2 = |V(G_2)|$. By Lemma 13, the (v_5, v_{k-4})-subpath of P is an induced (x_5, y_1)-path in G_2. By Lemma 10, there are at most $3^{(n_2-2)/3}$ such paths, and they can be enumerated in time $O^*(3^{n_2/3})$.

Since $D = D' \cup V(P)$, we obtain that there are at most $3^{n_1/3} \cdot 3^{(n_2-1)/3} \leq 3^{(n_1+n_2)/3} \leq 3^{n/3}$ minimal connected dominating sets D with the dominating pair $\{u, v\}$ in $H = G[D]$ and such that H has a shortest (u, v)-path $P = v_1 \dots v_k$ such that $v_i = x_i$ for $i \in \{1, \dots, 5\}$ and $v_i = y_{i+5-k}$ for $i \in \{k-4, \dots, k\}$. Moreover, these sets can be enumerated in time $O(3^{n/3})$. It follows that there are at most $3^{n/3} \cdot n^{10}$ minimal connected dominating sets D such that $H = G[D]$ has a dominating pair $\{u, v\}$ with $\mathrm{dist}_G(u, v) \geq 9$, and these sets can be enumerated in time $O(3^{n/3} \cdot n^{10})$.

We conclude that G has at most $3^{n/3} \cdot (n^{10} + n^9)$ minimal connected dominating sets that can be enumerated in time $O^*(3^{n/3})$. ◀

Proposition 1 implies that the bound for AT-free graphs is tight up to a polynomial factor.

6 Strongly chordal graphs

A vertex u of a graph G is *simple* if for any two neighbors x and y, $N_G[x] \subseteq N_G[y]$ or $N_G[y] \subseteq N_G[x]$. In other words, the closed neighborhoods of the neighbors of u are linearly ordered by inclusion. An ordering $v_1, \dots, v_n$ of $V(G)$ is a *simple elimination ordering* if for each $i \in \{1, \dots, n\}$, v_i is a simple vertex of $G[\{x_i, \dots, x_n\}]$.

▶ **Lemma 15** ([11]). *A graph is strongly chordal if and only if it has a simple elimination ordering.*

▶ **Theorem 16.** *A strongly chordal graph has at most $3^{n/3}$ minimal connected dominating sets, and these set can be enumerated in time $O^*(3^{n/3})$.*

Proof. We consider the following EnumCDS(H, X) algorithm that for a connected induced subgraph H of G and a set of vertices $X \subseteq V(G)$ enumerates the minimal connected dominating sets D of G such that $X \subseteq D$, $D \cap (V(G) \setminus V(H)) = X \cap (V(G) \setminus V(H))$ and $D \cap V(H)$ is a connected dominating set of H. This is a branching algorithm based on the property that any strongly chordal graph has a simple vertex by Lemma 15. If G is disconnected, then G has no connected dominating set. Assume that G is connected.

EnumCDS(H, X)

1. If $X \cap V(H)$ is a connected dominating set of H, then return X if X is a minimal connected dominating set of G and stop.
2. If H is a complete graph, then for each $v \in V(H)$, return $X \cup \{v\}$, and stop.
3. Consider a simple vertex $u \in V(H)$, and for each $v \in N_H(u)$, let $H' = H - (N_H[u] \setminus \{v\})$, $X' = X \cup \{v\}$ and call EnumCDS(H', X').

We call EnumCDS$(G, \emptyset)$ to enumerate minimal connected dominating sets of G.

The correctness of the algorithm is proved via the following claim, whose proof is given in the appendix.

Claim ($*$). *Suppose that D is a minimal connected dominating set of G, $X \subseteq D$ and H is a connected induced subgraph G such that*

- **(i)** *$D \cap (V(G) \setminus V(H)) = X \cap (V(G) \setminus V(H))$ and X dominates $V(G) \setminus V(H)$,*
- **(ii)** *any vertex w of H dominated by X in G is dominated by $X \cap V(H)$ in H and, moreover, if w is dominated by a vertex of a component F of $G[X]$, then w is dominated by $V(F) \cap V(H)$,*
- **(iii)** *for any component F of $G[X]$, $V(H) \cap V(F) \neq \emptyset$ and $G[V(F) \cap V(H)]$ is a component of $G[X \cap V(H)]$.*

Then

- **(a)** *if $X \cap V(H)$ is a connected dominating set of H, then $X = D$,*
- **(b)** *otherwise, if H is a clique, then $D = X \cup \{v\}$ for some $v \in V(H)$,*
- **(c)** *otherwise, if u is a simple vertex of H, then there is $v \in N_H(u)$ such that $X' = X \cup \{v\} \subseteq D$ and (i)–(iii) are fulfilled for $H' = H - (N_H[u] \setminus \{v\})$ and X'.*

Observe that the conditions (i)–(iii) of Claim ($*$) are fulfilled for $H = G$ and $X = \emptyset$. Applying Claim ($*$) recursively, we obtain that for any minimal connected dominating set D of G, EnumCDS$(G, \emptyset)$ outputs D at least once, i.e., EnumCDS$(G, \emptyset)$ enumerates the minimal connected dominating sets.

To obtain an upper bound for the number of minimal connected dominating sets, it is sufficient to upper bound the number of leaves in the search tree produced by EnumCDS. Notice that when we perform branching on Step 3 of EnumCDS for a simple vertex u with $d = d_H(u)$, we get d branches and in each branch we call EnumCDS for a graph with $|V(H)| - d$ vertices. By standard arguments (see, e.g., [16]), we obtain that the search tree for G has at most $3^{n/3}$ leaves. Hence, G has at most $3^{n/3}$ minimal connected dominating sets.

To complete the proof, notice that it is known that a simple elimination ordering of a strongly chordal graph can be found in polynomial time (see, e.g., [2, 4]). Observe also

that for any induced subgraph H of G, the ordering of its vertices induced by the simple elimination ordering for G is a simple elimination ordering. As each step of EnumCDS can be done in linear time, we conclude that EnumCDS runs in time $O^*(3^{n/3})$. ◀

As the class of interval graphs is a subclass of the strongly chordal graphs, the bound $3^{n/3}$ is tight by Proposition 1.

7 Distance-hereditary graphs

First we observe that the number of minimal connected dominating sets of a cograph is polynomial. The proof of Proposition 17 is given in the appendix.

▶ **Proposition 17.** *A cograph G with at least 3 vertices has at most $m = |E(G)|$ minimal connected dominating sets, and these can be enumerated in time $O(m)$.*

Notice that this bound is tight, e.g., for complete bipartite graphs. Now we consider distance-hereditary graphs. First, we need some additional terminology.

Let G be a connected graph and $u \in V(G)$. Denote by $L_0(u), \ldots, L_{s(u)}(u)$ the levels in the breadth-first search (BFS) of G starting at u. Hence for all $i \in \{0, \ldots, s(u)\}$, $L_i(u) = \{v \in V(G) \mid \mathrm{dist}_G(u, v) = i\}$. Clearly, the number of levels in this decomposition is $s(u) + 1$. For $i \in \{1, \ldots, s(u)\}$, we denote by $\mathcal{G}_i(u)$ the set of components of $G[L_i(u) \cup \ldots \cup L_{s(u)}(u)]$, and $\mathcal{G}(u) = \cup_{i=1}^{s(u)} \mathcal{G}_i(u)$. Let $H \in \mathcal{G}_i(u)$ and $B = N_G(V(H))$. Clearly, $B \subseteq L_{i-1}(u)$. We say that B is the *boundary of H in $L_{i-1}(u)$*. For $i \in \{0, \ldots, s(u) - 1\}$, $\mathcal{B}_i(u)$ is the set of boundaries in $L_i(u)$ of the graphs of $\mathcal{G}_{i+1}(u)$ and $\mathcal{B}(u) = \cup_{i=0}^{s(u)-1} \mathcal{B}_i(u)$.

▶ **Lemma 18** ([3, 10])**.** *A connected graph G is distance-hereditary if and only if for any vertex $u \in V(G)$, any $H \in \mathcal{G}(u)$ with the boundary B, the following holds: $N_G(u) \cap V(H) = N_G(v) \cap V(H)$ for $u, v \in B$.*

We also need the next observation that is implicit in [3, 10] but also can be easily proved directly.

▶ **Lemma 19** ([3, 10])**.** *Let G be a connected distance hereditary graph and $u \in V(G)$. Then for any $B_1, B_2 \in \mathcal{B}(u)$, either $B_1 \cap B_2 = \emptyset$ or $B_1 \subseteq B_2$ or $B_2 \subseteq B_1$.*

For a graph G and $u \in V(G)$, denote by $\mathcal{B}'(u)$ the set of inclusion minimal sets of $\mathcal{B}(u)$. Notice that by Lemma 19, the sets of $\mathcal{B}'(u)$ are pairwise disjoint if G is distance-hereditary. The enumeration of minimal connected dominating sets for distance-hereditary graphs is based on the following lemma.

▶ **Lemma 20.** *Let G be a distance hereditary graph with at least two vertices and $u \in V(G)$. Then for any minimal connected dominating set D with $u \in D$, $D \subseteq \cup_{B \in \mathcal{B}'(u)} B$ and $|B \cap D| = 1$ for $B \subseteq \mathcal{B}'(u)$.*

Proof. Let $H \in \mathcal{G}(u)$ and let $B \in \mathcal{B}(u)$ be its boundary. If $v \in V(H) \cap D \neq \emptyset$, then $B \cap D \neq \emptyset$, because $G[D]$ has an (u, v)-path and this path has a vertex of B. If $V(H) \cap D = \emptyset$, then D has a vertex v that dominates a vertex of H. Clearly, $v \in B$. We conclude that for each $B \in \mathcal{B}'(u)$, $|B \cap D| \geq 1$.

Since $n \geq 2$, $s(u) \geq 1$ and, therefore, $\mathcal{G}(u) \neq \emptyset$ and $\mathcal{B}'(u) \neq \emptyset$. In particular $\{u\} \in \mathcal{B}'(u)$. Recall that the sets of $\mathcal{B}'(u)$ are pairwise disjoint. Hence, there is a $D' \subseteq D$ such that $D \subseteq \cup_{B \in \mathcal{B}'(u)} B$ and $|B \cap D| = 1$ for $B \subseteq \mathcal{B}'(u)$. If $H \in \mathcal{G}_i(u)$ for $i \in \{1, \ldots, s(u)\}$, then any vertex of its boundary dominates $V(H) \cap L_i(u)$ by Lemma 18. Therefore, for each $H \in \mathcal{G}_i(u)$,

the vertices of $V(H) \cap L_i(u)$ are dominated. Hence, D' is a dominating set. Because for each $v \in D'$, $G[D']$ has a (u,v)-path by Lemma 18, D' is a connected dominating set. By minimality, we obtain that $D = D'$. ◀

▶ **Theorem 21.** *A distance-hereditary graph has at most $3^{n/3} \cdot n$ minimal connected dominating sets, and these sets can be enumerated in time $O^*(3^{n/3})$.*

Proof. If G is disconnected, it has no connected dominating set. If $n = 1$, then G has one connected dominating set and $1 \leq 3^{n/3} \cdot n$. Suppose that G is a connected graph and $n \geq 2$. For each $u \in V(G)$, G has at most $\prod_{B \in \mathcal{B}'(u)} |B|$ sets $D \subseteq \cup_{B \in \mathcal{B}'(u)} B$ such that $|D \cap B| = 1$ for $B \in \mathcal{B}'(u)$. By Lemma 20, G has at most $\prod_{B \in \mathcal{B}'(u)} |B|$ minimal connected dominating sets containing u. Because the sets of $\mathcal{B}'(u)$ are pairwise disjoint, $\sum_{B \in \mathcal{B}'(u)} |B| \leq n$. It is well known (see, e.g., [16]) that $\prod_{B \in \mathcal{B}'(u)} |B| \leq 3^{n/3}$ in this case. We obtain that the total number of minimal connected dominating sets is at most

$$\sum_{u \in V(G)} \prod_{B \in \mathcal{B}'(u)} |B| \leq 3^{n/3} \cdot n.$$

It is trivial to enumerate minimal connected dominating sets if G is disconnected or $n = 1$. To enumerate minimal connected dominating sets of a connected graph G with $n \geq 2$, we consider all possible choices of a vertex u. For each u, we perform the breadth-first search from u that can be done in linear time, and in time $O(nm)$ construct $\mathcal{B}'(u)$. Then the sets $D \subseteq \cup_{B \in \mathcal{B}'(u)} B$ such that $|D \cap B| = 1$ for $B \in \mathcal{B}'(u)$ can be enumerated in straightforward way in time $O^*(3^{n/3})$. Hence, the total running time is $O^*(3^{n/3})$. ◀

By Proposition 1 the obtained upper bound for distance-hereditary graphs is tight up to factor n.

8 Open questions

The most challenging question concernes the maximum number of minimal connected dominating sets in an n-vertex graph. No upper bound c^n with $c < 2$ is known; neither such an enumeration algorithm was achieved nor could it be achieved by combinatorics. The large gap between lower bound $3^{n/3}$ and upper bound 2^n is astonishing. Let us mention that it seems unlikely that the maximum number of minimal connected dominating sets in an n-vertex graph is upper bounded by $3^{n/3}$. If this were the case and the upper bound could be obtained by an enumeration algorithm, then this would drastically improve the running time of the best algorithm solving the minimum connected dominating set problem from $O(1.8619^n)$ to $O(1.4423^n)$.

References

1 Faisal N. Abu-Khzam, Amer E. Mouawad, and Mathieu Liedloff. An exact algorithm for connected red-blue dominating set. *J. Discrete Algorithms*, 9(3):252–262, 2011.

2 Richard P. Anstee and Martin Farber. Characterizations of totally balanced matrices. *J. Algorithms*, 5(2):215–230, 1984.

3 Hans-Jürgen Bandelt and Henry Martyn Mulder. Distance-hereditary graphs. *J. Comb. Theory, Ser. B*, 41(2):182–208, 1986.

4 Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph classes: a survey*. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1999.

5 Jesper Makholm Byskov. Enumerating maximal independent sets with applications to graph colouring. *Operations Research Letters*, 32(6):547–556, 2004.

6 Derek G. Corneil, Stephan Olariu, and Lorna Stewart. Asteroidal triple-free graphs. *SIAM J. Discrete Math.*, 10(3):399–430, 1997.

7 Jean-François Couturier, Pinar Heggernes, Pim van 't Hof, and Dieter Kratsch. Minimal dominating sets in graph classes: Combinatorial bounds and enumeration. *Theor. Comput. Sci.*, 487:82–94, 2013.

8 Jean-François Couturier, Pinar Heggernes, Pim van 't Hof, and Yngve Villanger. Maximum number of minimal feedback vertex sets in chordal graphs and cographs. In *COCOON 2012*, volume 7434 of *Lecture Notes in Computer Science*, pages 133–144. Springer, 2012.

9 Jean-François Couturier, Romain Letourneur, and Mathieu Liedloff. On the number of minimal dominating sets on some graph classes. *Theor. Comput. Sci.*, 562:634–642, 2015.

10 Alessandro D'Atri and Marina Moscarini. Distance-hereditary graphs, steiner trees, and connected domination. *SIAM J. Comput.*, 17(3):521–538, 1988.

11 Martin Farber. Characterizations of strongly chordal graphs. *Discrete Mathematics*, 43(2-3):173–189, 1983.

12 Fedor V. Fomin, Serge Gaspers, Artem V. Pyatkin, and Igor Razgon. On the minimum feedback vertex set problem: Exact and enumeration algorithms. *Algorithmica*, 52(2):293–307, 2008.

13 Fedor V. Fomin, Fabrizio Grandoni, Artem V. Pyatkin, and Alexey A. Stepanov. Combinatorial bounds via measure and conquer: Bounding minimal dominating sets and applications. *ACM Transactions on Algorithms*, 5(1), 2009.

14 Fedor V. Fomin, Pinar Heggernes, and Dieter Kratsch. Exact algorithms for graph homomorphisms. *Theor. Comp. Sys.*, 41(2):381–393, August 2007.

15 Fedor V. Fomin, Pinar Heggernes, Dieter Kratsch, Charis Papadopoulos, and Yngve Villanger. Enumerating minimal subset feedback vertex sets. *Algorithmica*, 69(1):216–231, 2014.

16 Fedor V. Fomin and Dieter Kratsch. *Exact exponential algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer, Heidelberg, 2010.

17 Fedor V. Fomin and Yngve Villanger. Finding induced subgraphs via minimal triangulations. In *STACS 2010*, volume 5 of *LIPIcs*, pages 383–394. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2010.

18 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.

19 Serge Gaspers and Simon Mackenzie. On the number of minimal separators in graphs. *CoRR*, abs/1503.01203, 2015.

20 Serge Gaspers and Matthias Mnich. Feedback vertex sets in tournaments. *Journal of Graph Theory*, 72(1):72–89, 2013.

21 Petr A. Golovach, Pinar Heggernes, Dieter Kratsch, and Reza Saei. Subset feedback vertex sets in chordal graphs. *J. Discrete Algorithms*, 26:7–15, 2014.

22 Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs*, volume 57 of *Annals of Discrete Mathematics*. Elsevier Science B.V., Amsterdam, second edition, 2004.

23 Pinar Heggernes. Minimal triangulations of graphs: A survey. *Discrete Mathematics*, 306(3):297–317, 2006.

24 Mamadou Moustapha Kanté, Vincent Limouzy, Arnaud Mary, and Lhouari Nourine. On the enumeration of minimal dominating sets and related notions. *SIAM J. Discrete Math.*, 28(4):1916–1929, 2014.

25 J. W. Moon and L. Moser. On cliques in graphs. *Israel J. Math.*, 3:23–28, 1965.

The Graph Motif Problem Parameterized by the Structure of the Input Graph

Édouard Bonnet[*1] **and Florian Sikora**[2]

1 **Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), Budapest, Hungary**
bonnet.edouard@sztaki.mta.hu

2 **PSL, Université Paris-Dauphine, LAMSADE UMR CNRS 7243, France**
florian.sikora@dauphine.fr

Abstract

The Graph Motif problem was introduced in 2006 in the context of biological networks. It consists of deciding whether or not a multiset of colors occurs in a connected subgraph of a vertex-colored graph. Graph Motif has been analyzed from the standpoint of parameterized complexity. The main parameters which came into consideration were the size of the multiset and the number of colors. Though, in the many applications of Graph Motif, the input graph originates from real-life and has structure. Motivated by this prosaic observation, we systematically study its complexity relatively to graph structural parameters. For a wide range of parameters, we give new or improved FPT algorithms, or show that the problem remains intractable. Interestingly, we establish that Graph Motif is W[1]-hard (while in W[P]) for parameter max leaf number, which is, to the best of our knowledge, the first problem to behave this way.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Parameterized Complexity, Structural Parameters, Graph Motif, Computational Biology

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.319

1 Introduction

The Graph Motif problem has received a lot of attention during the last decade. Informally, Graph Motif is defined as follows: given a graph with arbitrary colors on the nodes and a multiset of colors called the motif, the goal is to decide if there exists a subset of vertices of the graph such that (1) the subgraph induced by this subset is connected and (2) the colors on the subset of vertices match the motif, i.e. each color appears the same number of times as in the motif. Originally, this problem is motivated by applications in biological network analysis [24]. However, it proves useful in social or technical networks [4] or in the context of mass spectrometry [8].

Studying biological networks allows a better characterization of species, by determining small recurring subnetworks, often called *motifs*. Such motifs can correspond to a set of nodes realizing some function, which may have been evolutionary preserved. Thus, it is crucial to determine these motifs to identify common elements between species and transfer the biological knowledge. Graph Motif corresponds to topology-free queries and can be

* This work is partially supported by ERC Starting Grant PARAMTIGHT (n. 280152).

seen as a variant of a graph pattern matching problem with the sole topological requirement of connectedness. Such queries were also studied extensively for sequences during the last thirty years, and with the increase of knowledge about biological networks, it is relevant to extend these queries to networks [30].

2 Preliminaries and previous work

For any two integers $x < y$, we set $[x, y] := \{x, x+1, \ldots, y-1, y\}$, and for any positive integer x, $[x] := [1, x]$. If $G = (V, E)$ is a graph and $S \subseteq V$ a subset of vertices, $G[S]$ denotes the subgraph of G induced by S. For a vertex $v \in V$, the set of neighbors of v in G is denoted by $N_G(v)$, or simply $N(v)$, and $N_G(S) := (\bigcup_{v \in S} N(v)) \setminus S$ and will often be written just $N(S)$. We define $N[v] := N(v) \cup \{v\}$ and $N[S] := N(S) \cup S$. We say that a vertex v *dominates* a set of vertices S if $S \subseteq N[v]$. A set of vertices R *dominates* another set of vertices S if $S \subseteq N[R]$. If $G = (V, E)$ is a graph and $V' \subseteq V$, $G - V'$ denotes the graph $G[V \setminus V']$. A *universal vertex* v, in a graph $G = (V, E)$, is such that $N_G[v] = V$. A *matching* of a graph is a mutually disjoint set of edges. In an explicitly bipartite graph $G = (V_1 \cup V_2, E)$, we call a matching of size $\min(|V_1|, |V_2|)$ a *perfect matching*. A *cluster graph* (or simply, *cluster*) is a disjoint union of cliques. A *co-cluster graph* (or, *co-cluster*) is the complement graph of a cluster graph. If $\mathcal{C}$ is a class of graphs, the *distance to* $\mathcal{C}$ of a graph G is the minimum number of vertices to remove from G to get a graph in $\mathcal{C}$.

If $f : A \to B$ is a function and $A' \subseteq A$, $f_{|A'}$ denotes the restriction of f to A', that is $f_{|A'} : A' \to B$ such that $\forall x \in A'$, $f_{|A'}(x) := f(x)$. Similarly, if E is a set of edges on vertices of V and $V' \subseteq V$, $E_{|V'}$ is the subset of edges of E having both endpoints in V'.

Graph Motif and multisets. Graph Motif is defined as follows:

Graph Motif

- **Input**: A triple (G, c, M), where $G = (V, E)$ is a graph, $c : V \to \mathcal{C}$ gives some color of $|\mathcal{C}|$ to the vertices, and M is a multiset of colors of $\mathcal{C}$.
- **Output**: A subset $P \subseteq V$ such that (1) $G[P]$ is connected and (2) $c(P) = M$.

We will refer to condition (1) as the *connectivity constraint* and to condition (2) as the *multiset constraint*. For convenience, if $S \subseteq V$, $c(S)$ will denote the multiset of colors of vertices in S.

The *multiplicity* of element x in multiset M, denoted by $m_M(x)$ is the number of occurences of x in M. The cardinality of a multiset M denoted by $|M|$ is its number of elements *with their multiplicity*: $\Sigma_{x \in M} m_M(x)$. If M and N are two multisets, $M \cup N$ is the multiset A such that $\forall x$, $m_A(x) = m_M(x) + m_N(x)$, and $M \setminus N$ is the multiset D such that $\forall x \in M$, $m_D(x) = \max(0, m_M(x) - m_N(x))$ (and $\forall x \notin M$, $m_D(x) = 0$). We write $M \subseteq N$ iff $M \setminus N = \emptyset$ and $M \subset N$ iff $M \subseteq N$ and $M \neq N$. For example, let $M = \{1, 2, 2, 4, 5, 5, 5\}$ and $N = \{1, 1, 1, 2, 2, 3, 3, 4, 5, 5, 5, 5\}$. Here, $|M| = 7$, $|N| = 12$, $M \setminus N = \emptyset$, $N \setminus M = \{1, 1, 3, 3, 5\}$, and $M \subseteq N$.

Parameterized Complexity and ETH. A parameterized problem (I, k) is said *fixed-parameter tractable* (or in the class FPT) w.r.t. (with respect to) parameter k if it can be solved in $f(k) \cdot |I|^c$ time (in *fpt-time*), where f is any computable function and c is a constant (see [28, 14] for more details about fixed-parameter tractability). The parameterized complexity hierarchy is composed of the classes $\mathsf{FPT} \subseteq \mathsf{W}[1] \subseteq \mathsf{W}[2] \subseteq \cdots \subseteq \mathsf{W}[\mathsf{P}] \subseteq \mathsf{XP}$. The class XP contains problems solvable in time $|I|^{f(k)}$, where f is an unrestricted function.

A powerful technique to design parameterized algorithms is *kernelization*. In short, kernelization is a polynomial-time self-reduction algorithm that takes an instance (I, k) of a parameterized problem P as input and computes an equivalent instance (I', k') of P such that $|I'| \leqslant h(k)$ for some computable function h and $k' \leqslant k$. The instance (I', k') is called a *kernel* in this case. If the function h is polynomial, we say that (I', k') is a polynomial kernel.

The *Exponential Time Hypothesis* (ETH) is a conjecture by Impagliazzo et al. [21] asserting that there is no $2^{o(n)}$-time algorithm for 3-SAT on instances with n variables. The so-called sparsification lemma, also proved in [21], shows that if ETH turns out to be true, then there is no $2^{o(n+m)}$-time algorithm solving 3-SAT where m is the number of clauses.

Previous work. Many results about the complexity of Graph Motif are known. The problem is NP-hard even with strong restrictions. For instance, it remains NP-hard for bipartite graphs of maximum degree 4 and motifs containing two colors only [15], or for trees of maximum degree 3 and when the motif is colorful (that is, no color occurs more than once) [15], or for rooted trees of depth 2 [2]. However, the problem is solvable in polynomial time when the graph is a caterpillar [2], or when both the number of colors in the motif and the treewidth of the graph are bounded by a constant [15].

As Graph Motif is intractable even for very restricted classes of graphs, and considering that, in practice, the motif is supposed to be small compared to the graph, the parameterized complexity of Graph Motif relatively to the size of the motif has been tackled. It is indeed in FPT when parameterized by the size of the motif. At least seven different papers gave an FPT algorithm [15, 4, 20, 23, 5, 30, 29]. The best (randomized) algorithm runs in time $O^*(2^k)$ where the O^* notation suppresses polynomial factors [5, 30] and works well in practice for small values of k, even with hundreds of millions of edges [6]. The current best deterministic algorithm takes time $O^*(5.22^k)$ [29]. However, an algorithm running in time $O^*((2-\epsilon)^k)$ would break the 2^n barrier in solving Set Cover instances with n elements [5]. Besides, it is unlikely that Graph Motif admits a polynomial kernel, even on a restricted class of trees [2]. Ganian also proved that the problem is in FPT when the parameter is the size of a minimum vertex cover of the graph [17]. Actually, his algorithm is given for a smaller parameter called twin-cover. Ganian also show that Graph Motif can be solved in $O^*(2^k)$ for graphs with neighborhood diversity k [18]. On the negative side, the problem is W[1]-hard relatively to the number of colors, even for trees [15]. To deal with the huge rate of noise in the biological data, many variants of the problem has been introduced. For example, the approach of Dondi *et al.* requires a solution with a minimum number of connected components [13], while the one of Betzler *et al.* asks for a 2-connected solution [4]. In other variants stemming purely from bio-informatics, some colors can be added to, substituted or subtracted from the solution [10, 13].

In light of the previous paragraphs, it is clear that the complexity of Graph Motif is well known for different versions and constraints on the problem itself. However, only few works take into account the structure of the input graph. We believe that this an interesting direction since Graph Motif has applications in real-life problems, where the input is not random. For example, some biological networks have been shown scale-free or with small diameter [1]. We will therefore introduce a systematic study with respect to structural graph parameters [22, 16]. We believe that this is also of theoretical interest, to understand how a given parameter influences the complexity of the problem.

Organization. In Section 3, we improve the known FPT algorithms with parameter distance to clique, vertex cover number, and edge clique cover number. We also give a parameterized

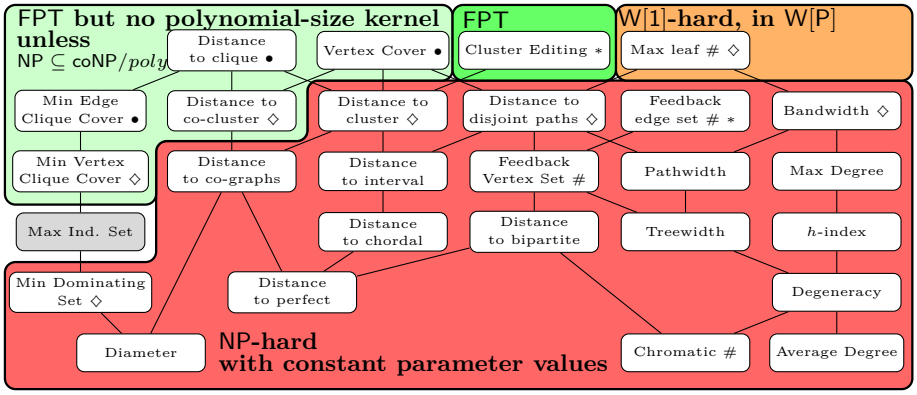


Figure 1 Hasse diagram of the relationship between different parameters ([22]). Two parameters are connected by a line if the parameter below can be polynomially upper-bounded in the parameter above. For example, *vertex cover* is above *distance to disjoint paths* since deleting a vertex cover produces an independent set, hence a set of disjoint paths. Therefore, positive results propagate upwards, while negative results propagate downwards. Results marked by ♢ are obtained in this paper, those marked with • are improvement of existing results, and those marked with ∗ are corollaries of existing results.

algorithm for the parameter distance to co-cluster which nicely reuses the FPT algorithms for both vertex cover number and distance to clique and another algorithm for parameter vertex clique cover number. These last two algorithms are noteworthy since a bounded distance to co-cluster or a bounded vertex clique cover number do not imply a bounded neighborhood diversity, a parameter for which Graph Motif was already known to be in FPT. We also show that a polynomial kernel for the aforementioned parameters is unlikely. In Section 4, we show that Graph Motif remains hard on graphs of constant distance to disjoint paths, or constant bandwidth, or constant distance to cluster, or constant dominating set number. More surprisingly, we establish that Graph Motif is W[1]-hard (but in W[P]) for the parameter max leaf number. To the best of our knowledge, there is no previously known problem behaving similarly when parameterized by max leaf number. Indeed, graphs with bounded max leaf number are really simple and, for instance, all the problems studied in [16] are FPT for this parameter. These positive and negative results draw a tight line between tractability and intractability (see Figure 1). Due to space constraints, some proofs (marked with ★) are deferred to the full version of the paper.

3 FTP algorithms and lower bound in the size of kernels

In this section, we improve or establish new FPT algorithms for several parameters. We also give a lower bound on the size of the kernel for all those parameters except *cluster editing number*. Figure 1 summarizes those results.

3.1 Cluster editing and linear neighborhood diversity

The cluster editing number of a graph is the number of edge deletions or additions required to get a cluster graph. It can be computed in time $O^*(1.62^k)$ [7]. We will use a known result involving another parameter called neighborhood diversity introduced by Lampis [25]. A graph has neighborhood diversity k if there is a partition of its vertices into at most k sets

such that all the vertices in each set *have the same type*. And, two vertices u and v *have the same type* iff $N(v) \setminus \{u\} = N(u) \setminus \{v\}$. We say that a graph parameter κ has *linear* (resp. *exponential*) *neighborhood diversity* if, for every positive integer k, all the graphs G such that $\kappa(G) \leqslant k$ have neighborhood diversity ck (resp. c^k) for some constant c. We say that a parameter κ has *unbounded neighborhood diversity*, if there is *no* function f such that all graphs G with $\kappa(G) \leqslant k$ have neighborhood diversity $f(k)$.

▶ **Theorem 1** ([18]). Graph Motif *can be solved in* $O^*(2^k)$ *on graphs with neighborhood diversity* k.

The following result is a direct consequence of the fact that, restricted to connected graphs, cluster editing has linear neighborhood diversity.

▶ **Corollary 2.** Graph Motif *can be solved in* $O^*(8^k)$*, where* k *is the cluster editing number.*

Proof. Let $(G = (V, E), c, M)$ be any instance of Graph Motif. We can assume that G is connected, otherwise we run the algorithm in each connected component of G. Let X be the set of vertices which are an endpoint of an edited edge (deleted or added) and let G' be the cluster graph obtained by the k edge editions. We may observe that $|X| \leqslant 2k$ and that the number of maximal cliques $C_1, \ldots, C_l$ in G' is bounded by k (otherwise, G could not be connected). For each $i \in [l]$, and for each vertex $v \in C_i \setminus X$, $N[x] = C_i$. Thus the neighborhood diversity of G is bounded by $|X| + l \leqslant 2k + k = 3k$. So, we can run the algorithm for bounded neighborhood diversity [18] and it takes time $O^*(2^{3k})$. ◀

3.2 Parameters with exponential neighborhood diversity

The next three parameters that we consider are *distance to clique*, *size of a minimum vertex cover*, and *size of a minimum edge clique cover*. For the first two, a value of k entails that the neighborhood diversity is at most $k + 2^k$; and neighborhood diversity 2^k for the third one. Therefore, Ganian has already given an algorithm running in double exponential time for these parameters ($O^*(2^{k+2^k})$ or $O^*(2^{2^k})$, see Theorem 1, [17, 18]). We improve this bound to single exponential time $2^{O(k)}$ (more precisely $O^*(8^k)$) for distance to clique and to $2^{O(k \log k)}$ for the vertex cover and edge clique cover numbers. The latter running time is sometimes called *slightly superexponential* FPT time [26]. Then, we prove that for each of those three parameters, a polynomial kernel is unlikely.

As a preparatory lemma for the algorithm parameterized by distance to clique, we show that a variant of Set Cover with thresholds is solvable in time $O^*(2^n)$, where n is the size of the universe. In the problem that we call here Colored Set Cover with Thresholds, one is given a triple $(\mathcal{U}, \mathcal{S} = \mathcal{C}_1 \uplus \ldots \uplus \mathcal{C}_l, (a_1, \ldots, a_l))$ where $\mathcal{U}$ is a ground set of n elements, $\mathcal{S}$ is a set of subsets of $\mathcal{U}$ partitioned into l classes called *colors* and $(a_1, \ldots, a_l)$ is a tuple of l positive integers called *threshold vector*. The goal is to find a set cover $\mathcal{T} \subseteq \mathcal{S}$ (not necessarily minimum) such that for each $i \in [l]$, the number of sets with color i (that is, in $\mathcal{C}_i$) in $\mathcal{T}$ is at most a_i.

▶ **Lemma 3.** Colored Set Cover with Thresholds *with* n *elements and* m *sets can be solved in time* $O(nm2^n + nm)$.

Proof. We order the sets of $\mathcal{S}$ such that sets of the same color appear consecutively, say, first the sets of $\mathcal{C}_1$, then the sets of $\mathcal{C}_2$, and so on. The order within the sets of a same color is not important and is chosen arbitrarily. We denote the sets resultantly ordered by $S_1, \ldots, S_m$ and function c maps the index of a set to its color. Therefore, $c(j) = i$ means that set S_j has

color i ($S_j \in C_i$). We fill by dynamic programming the table T, where $T[U, j]$ is meant to contain the minimum number of sets in $\mathcal{C}_{c(j)}$ among any subset of $\{S_1, \ldots, S_j\}$ that covers $U \subseteq \mathcal{U}$ and respects the threshold vector.

As an initialization step, for each $U \subseteq \mathcal{U}$, we set $T[U, 1] = 1$ if $U \subseteq S_1$, and $T[U, 1] = \infty$ otherwise. For each $j \in [2, m]$, assuming that $T[U', j-1]$ was already filled for every $U' \subseteq \mathcal{U}$, we distinguish two cases to fill $T[U, j]$. If S_j is the first set of the color class $\mathcal{C}_{c(j)}$ then:

$$T[U,j] = \begin{cases} 0 \text{ if } T[U,j-1] < \infty & \text{(* discard } S_j \text{ *)} \\ 1 \text{ if } T[U,j-1] = \infty \text{ and } T[U \setminus S_j, j-1] < \infty & \text{(* add } S_j \text{ *)} \\ \infty \text{ otherwise} \end{cases}$$

Otherwise S_j is not the first set in $\mathcal{C}_{c(j)}$ and:

$$T[U,j] = \min \begin{cases} T[U,j-1] & \text{(* discard } S_j \text{ *)} \\ v+1 \text{ if } v < a_{c(j)} \text{ and } \infty \text{ otherwise} & \text{(* add } S_j \text{ *)} \end{cases}$$

with $v = T[U \setminus S_j, j-1]$.

A standard induction shows that the instance is positive iff $T[\mathcal{U}, m] \neq \infty$. The only costly operation in filling one entry of table T is the set difference which can be done in $O(n)$. If we want to produce an actual solution (and not solely decide the problem), we can add one bit in each entry $T[U, j]$ signaling whether or not S_j should be taken. Should the instance be positive, it then takes time $O(nm)$ to reconstruct a solution from a filled table T. Therefore, the running time is $O(n|T| + nm) = O(nm2^n + nm)$. ◀

▶ **Theorem 4.** Graph Motif *can be solved in* $O^*(8^k)$*, where* k *is the distance to clique.*

Proof. Let $(G = (V, E), c : V \to \mathcal{C}, M)$ be any instance of Graph Motif and assume R is a solution, that is $G[R]$ is connected and $c(R) = M$. If there is no solution, our algorithm will detect it eventually. We first compute a set $S \subseteq V$ of size k such that $C := V \setminus S$ is a clique. This can be done in time $O^*(2^k)$ by branching over the two endpoints of a *non-edge*, or even in $O^*(1.2738^k)$ by applying the state-of-the-art algorithm for Vertex Cover on the complementary graph [11]. Running through all the 2^k subsets of S, one can guess the subset $S' = R \cap S$ of S which is in the solution R, and $S_1, S_2, \ldots, S_{k'}$ be the $k' \leqslant k$ connected components of $G[S']$. It must hold that $c(S') \subseteq M$, otherwise R would not be a solution. Now, the problem boils down to finding a non-empty (an empty subset would mean that $S' = R$ which can be easily checked) subset $C' \subseteq C$ such that $G[S' \cup C']$ is connected and $c(C') \subseteq M \setminus c(S')$. Then, the set $S' \cup C'$ can be extended into a solution by adding vertices of $C \setminus C'$ with the right colors. The graph $G[S' \cup C']$ is connected iff each connected component S_j of $G[S']$ has at least one neighbor in $N(C')$. We build an equivalent instance of Colored Set Cover with Thresholds in the following way. The ground set $\mathcal{U}$ is of size k' with one element x_j per connected component S_j of $G[S']$. For each vertex v in C colored by i, there is a set S_v colored by i such that $x_j \in S_v$ iff $N(v) \cap S_j \neq \emptyset$. For each color i, the threshold a_i is set to the multiplicity of i in $M \setminus c(S')$. If there are more than one set with the same color and the same elements, we keep only one copy of this colored set. The number of sets is therefore at most $2^{k'}|\mathcal{C}|$. So, it takes time $O(k'2^{k'}|\mathcal{C}|(2^{k'}+1)) = O^*(4^{k'})$ to solve this instance, hence an overall worst case running time of $O^*(2^k + 2^k 4^k) = O^*(8^k)$. ◀

▶ **Theorem 5.** Graph Motif *can be solved in* $O^*(2^{2k \log k})$ *on graphs with a vertex cover of size* k*.*

Proof. We start similarly to the previous algorithm. We compute a minimum vertex cover S of G in time $O^*(2^k)$ (or $O^*(1.2738^k)$ [11]), and then guess in time $O^*(2^k)$ the subset $S' = S \cap R$, where R is a fixed solution. Again, we denote by $S_1, S_2, \ldots, S_{k'}$ the connected components of $G[S']$. We remove $c(S')$ from the motif and we remove from V the set I'

of the vertices of the independent set $I := V \setminus C$ which have no neighbor in S'. Now, by the transformation presented in the algorithm parameterized by distance to clique, the problem could be made equivalent to a constrained version of Colored Set Cover with Thresholds where the intersection graph (with an edge between two sets if they have a non-empty intersection) of the solution has to be connected. Unfortunately, it is not clear whether or not this variant can be solved in time $2^{O(n)}$. Thus, at this point, we have to do something different.

Let $R_d = \{r_1, r_2, \ldots, r_l\} \subseteq R \setminus S'$ be a minimal (inclusion-wise) set of vertices such that $G[S' \cup R_d]$ is connected. We can observe that $l \leqslant k' \leqslant k$. We guess in time $O^*(l!B_l)$ (where B_l is the l-th Bell number, i.e., the number of partitions of a set of size l) an ordered partition $P := \langle A_1, A_2, \ldots, A_l \rangle$ of the connected components $\{S_1, \ldots, S_{k'}\}$ such that, for each $i \in [l]$, (1) r_i has at least one neighbor in each connected component of A_i and (2) if $i \geqslant 2$, r_i has at least one neighbor in a connected component of $\bigcup_{1 \leqslant j < i} A_j$. Note that such an ordered partition always exists since $G[S' \cup R_d]$ is connected. Now, we build the bipartite graph $B = (P \cup M', F)$, where $M' = M \setminus c(S')$ and there is an edge between $A_i \in P$ and each copy of color $c \in M'$ iff there is a vertex $v \in I$ colored by c in the original graph G and such that (1) v has at least one neighbor in each connected component of A_i and (2) if $i \geqslant 2$, v has at least one neighbor in a connected component of $\bigcup_{1 \leqslant j < i} A_j$. By construction, $\{\{A_i, c(r_i)\} \mid i \in [l]\}$ is a maximum matching of size $|P| = l$ in graph B. Thus, we compute in polynomial time a maximum matching $\{\{A_i, c_i\} \mid i \in [l]\}$ in B. Then, we obtain a solution to the Graph Motif instance by taking, for each $i \in |l]$ any vertex v_i colored by c_i and having (1) at least one neighbor in each connected component of A_i and (2) if $i \geqslant 2$, at least one neighbor in a connected component of $\bigcup_{1 \leqslant j < i} A_j$. This can also be done in polynomial time and the existence of such a v_i is guaranteed by the construction of graph B. Then, we complete set $S' \cup \bigcup_{i \in [l]} \{v_i\}$ into a solution by taking any vertices in $I \setminus I'$ with the right colors. As $l! \leqslant l^l$, $B_l \leqslant (\frac{l}{2})^l$ (even $B_l < (\frac{0.792l}{\ln(l+1)})^l$ [3]), and $l \leqslant k$ the overall running time is $O^*(2^k + 2^k k! B_k) = O^*(k^k k^k) = O^*(2^{2k \log k})$. ◀

In the Edge Clique Cover problem, one asks, given a graph $G = (V, E)$ and an integer k, for k subsets $C_1, \ldots, C_k \subseteq V$, such that $\forall i \in [k]$, $G[C_i]$ is a clique, and $\forall e \in E$, e lies in a clique C_i for some $i \in [k]$. The set $\{C_1, \ldots, C_k\}$ is called an *edge clique cover* of G. The *edge clique cover number* of a graph G is the smallest k such that G has an edge clique cover of size k. Edge Clique Cover admits a kernel of size 2^k [19] and, as observed in [12], it can be solved by dynamic programming in time $2^{O(n+m)}$. Therefore, it can be solved in time $2^{O(2^k + 2^{2k})}$, that is $2^{2^{O(k)}}$. On the negative side, Edge Clique Cover cannot be solved in time $2^{2^{o(k)}}$ under ETH [12]. Thus, the algorithm of Ganian [18] is essentially optimal if the edge clique cover is not given. But, we may imagine that the instance comes with an optimal or close to optimal edge clique cover, or that we have a good heuristic to compute it (a polynomial time approximation with sufficiently good ratio is unlikely [27]).

▶ **Theorem 6.** Graph Motif *can be solved in time* $2^{2^{O(k)}}$*, where* k *is the edge clique cover number, and in time* $O^*(2^{2k \log k + k})$ *if an edge clique cover of size* k *is given as part of the input.*

Proof. Let $(G = (V, E), c, M)$ be any instance of Graph Motif. If not given, we first compute an edge clique cover $\{C_1, \ldots, C_k\}$ of size k in G, in time $2^{2^{O(k)}}$ [19]. We guess in time $O^*(2^k)$ the exact subset $\{C'_1, \ldots, C'_{k'}\} \subseteq \{C_1, \ldots, C_k\}$ of cliques C_i such that $C_i \cap R$ is non-empty, for a fixed solution R. Now, we turn the instance into an equivalent instance where the motif has size $|M| + k'$ and the graph has at most $|V| + k'$ vertices and a vertex cover of size k'. The new graph is a bipartite graph $B = (A \cup W, F)$ such that A contains one vertex

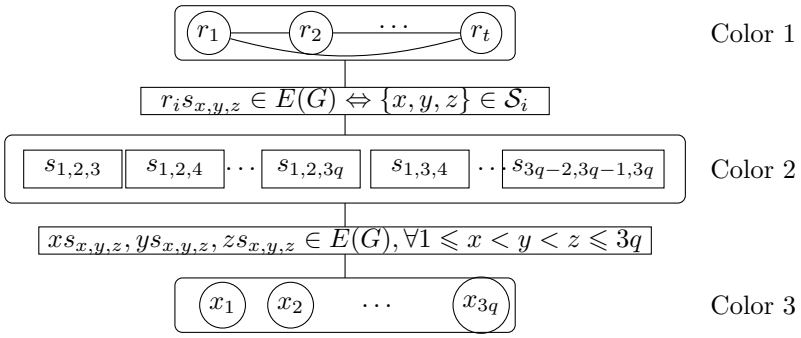


Figure 2 Illustration of the construction of G. The motif consists of 1 occurrence of color 1, q of color 2 and $3q$ of color 3.

$v(C'_i)$ per clique C'_i (so, A is a vertex cover of graph B of size $k' \leqslant k$), $W = C'_1 \cup \ldots \cup C'_{k'} \subseteq V$, and there is an edge in F between $v(C'_i) \in A$ and $w \in W$ iff $w \in C'_i$. Each vertex in W keeps the color it had in G. A fresh color c is given to the k' vertices of A, and color c is added to the motif M with multiplicity k'. Then, we run the algorithm parameterized by the vertex cover number of Theorem 5. This algorithm has an overall running time of $O^*(2^k 2^{2k \log k})$, if the edge clique cover is given, and $2^{2^{O(k)}}$ otherwise. ◀

Ganian [17], Theorem 5 and Theorem 4 prove that Graph Motif is in FPT if the parameter is the vertex cover number or the distance to clique. Therefore, the problem has a kernel [28]. Though, the size of this kernel is *a priori* not known. We show that the corresponding kernels cannot be polynomial unless NP $\subseteq$ coNP/*poly*.

▶ **Theorem 7.** *Unless* NP $\subseteq$ coNP/*poly*, Graph Motif *has no polynomial kernel when parameterized by the vertex cover number or the distance to clique, even for* (i) *motifs with only 3 colors and* (ii) *when the motif is colorful.*

Proof. We only give the proof for (i). The second item (ii) can be proven similarly following the ideas of [5].

We will define an OR-cross-composition [9] from the NP-complete X3C problem, stated as follows: given and integer q, a set $X = \{x_1, x_2, \ldots, x_{3q}\}$ and a collection $\mathcal{S} = \{S_1, \ldots, S_{|\mathcal{S}|}\}$ of 3-elements subsets of X, the goal is to decide if $\mathcal{S}$ contains a subcollection $\mathcal{T} \subseteq \mathcal{S}$ such that $|\mathcal{T}| = q$ and each element of X occurs in exactly one element of $\mathcal{T}$. Given t instances, $(X_1, \mathcal{S}_1), (X_2, \mathcal{S}_2), \ldots, (X_t, \mathcal{S}_t)$, of X3C, we define our equivalence relation $\mathcal{R}$ such that any strings that are not encoding valid instances are equivalent, and $(X_i, \mathcal{S}_i), (X_j, \mathcal{S}_j)$ are equivalent iff $|X_i| = |X_j|$ and $|\mathcal{S}_i| = |\mathcal{S}_j|$. Hereafter, we assume that $X_i = [3q]$ and $\mathcal{S}_i = \{S_1, \ldots, S_{|\mathcal{S}_i|}\}$, for any $i \in [t]$. We will build an instance (G, c, M) of Graph Motif parameterized by the vertex cover or the distance to clique, where G is the input graph, c the coloring function and M the motif, such that there is a solution for Graph Motif iff there is an $i \in [t]$ such that there is a solution for $(X_i, \mathcal{S}_i)$. We will now describe how to build such instance of Graph Motif. The graph G consists of t nodes $r_1, r_2, \cdots, r_t$ forming a clique. There are also $O((3q)^3)$ nodes $s_{x,y,z}, 1 \leqslant x < y < z \leqslant 3q$, with an edge between r_i and $s_{x,y,z}$ iff the 3-element subset $\{x, y, z\}$ exists in $\mathcal{S}_i$. Finally, there are $3q$ nodes $x_i, 1 \leqslant i \leqslant 3q$, and there is an edge between x_i and every subset $s_{x,y,z}$ where x_i occurs (see also Figure 2). The coloration is $c(r_i) = 1$, for all $1 \leqslant i \leqslant t$, $c(s_{x,y,z}) = 2$ for all $1 \leqslant x < y < z \leqslant 3q$, and $c(x_i) = 3, 1 \leqslant i \leqslant 3q$. The multiset M consists of 1 occurrence of the color 1, q occurrences of color 2 and $3q$ occurrences of color 3.

It is easy to see that $\{s_{x,y,z} | 1 \leqslant x < y < z \leqslant 3q\} \cup \{x_i | 1 \leqslant i \leqslant 3q\}$ is a vertex cover for G and that its removal leaves only a clique, and that its size is polynomial in $3q$ and hence in the size of the largest instance.

Let us show that there is a solution for our instance of Graph Motif iff at least one of the $(X_i, \mathcal{S}_i)$'s has a solution of size q.

($\Leftarrow$) Suppose that $(X_i, \mathcal{S}_i)$ has a solution $\mathcal{T}_i$ of size q. We set $P = \{r_i\} \cup \{s_{x,y,z} \mid \{x, y, z\} \in \mathcal{T}_i\} \cup \{x_i | 1 \leqslant i \leqslant 3q\}$. One can easily check that $G[P]$ is connected and that $c(P) = M$.

($\Rightarrow$) Suppose that there is a solution $P \subseteq V$ such that $G[P]$ is connected and $c(P) = M$. Due to the motif, only one of the nodes r_i is in P and all nodes x_i are in P. We claim that there is then a solution $\mathcal{T}_i$ in $(X_i, \mathcal{S}_i)$, where i is the index of the only node r_i in P. We add in $\mathcal{T}_i$ the q sets $\{x, y, z\}$ such that $s_{x,y,z} \in P$. By the connectivity constraint, these sets all occurs in the instance i s.t. $r_i \in P$. Let us now prove that $\mathcal{T}_i$ covers exactly all the elements of X_i. Since P is a solution, the nodes $s_{x,y,z}$ in P correspond to a partition of X. Otherwise, one of the node x_i will not be connected. ◀

3.3 Parameters with unbounded neighborhood diversity

This section disproves the idea that Graph Motif is only tractable for classes with bounded neighborhood diversity. Indeed, we show that Graph Motif is in FPT parameterized by the size of a *vertex clique cover* or by the distance to co-cluster. The former algorithm creates a win/win based on König's theorem applied to a bounded number of auxiliary bipartite graphs. The latter is simpler and use as subroutines the algorithms parameterized by vertex cover number and distance to clique.

In the Vertex Clique Cover problem (also known as Clique Partition), one asks, given a graph $G = (V, E)$ and an integer k, for a *partition* of the vertices into k subsets $C_1, \ldots, C_k \subseteq V$, such that $\forall i \in [k]$, $G[C_i]$ is a clique. The set $\{C_1, \ldots, C_k\}$ is called an *vertex clique cover* of G. The *vertex clique cover number* of a graph G is the smallest k such that G has an vertex clique cover of size k. This problem is equivalent to the Graph Coloring problem since a graph as a vertex clique cover of size k iff its complement is k-colorable. Therefore, Vertex Clique Cover is unlikely to be in XP. However, if a vertex clique cover comes with the input, we show that Graph Motif is in FPT for parameter vertex clique cover number. One can notice that Graph Motif is NP-hard in 2-colorable graphs. This is a striking example of how easier can Graph Motif be on the denser counterpart of two complementary classes.

To realize that vertex clique cover number has unbounded neighborhood diversity, think of the complement of a bipartite graph. The vertex clique cover is of size 2 but the neighborhood diversity could be arbitrary; for parameter distance to co-cluster, think of the complementary of a cluster graph with an unbounded number of cliques.

▶ **Theorem 8 (★).** Graph Motif *can be solved in time* $O^*(2^{4k \log(2k)})$ *where* k *is the vertex clique cover number, provided that the vertex clique cover is given as part of the input.*

▶ **Theorem 9 (★).** Graph Motif *can be solved in* $O^*(2^{2k \log k})$*, where* k *is the distance to co-cluster.*

4 Parameters for which Graph Motif is hard

In this section, we provide several parameters for which Graph Motif is not in XP, unless P = NP. In other words, the problem is NP-hard even for fixed values of the parameter. We

also prove that the problem remains W[1]-hard for parameter max leaf number. Figure 1 summarizes these results.

4.1 Deletion set numbers

We study parameters which correspond to the minimum number of vertices to remove to make the graph belong to a restricted class. We will show that Graph Motif remains NP-hard for constant values of those parameters. More precisely, the colorful restriction of Graph Motif is hard even if we can obtain a set of disjoint paths by removing 1 vertex, a cluster graph by removing 1 vertex, and an acyclic graph by removing 0 edge.

▶ **Theorem 10** ([15]). Graph Motif *is* NP*-hard even when G is a tree of maximum degree 3 and the motif is colorful.*

▶ **Corollary 11.** Graph Motif *is* NP*-hard even for graphs with feedback edge set 0 and when the motif is colorful.*

▶ **Theorem 12 (★).** Graph Motif *is* NP*-hard even (i) for graphs with distance* 1 *to disjoint paths and when the motif is colorful and (ii) for graphs with bandwidth* 4 *and when the motif is colorful.*

▶ **Theorem 13 (★).** Graph Motif *is* NP*-hard even for graphs with distance* 1 *to cluster and when the motif is colorful.*

4.2 Dominating set number

Being given a small dominating set of the graph cannot help in solving Graph Motif. For any instance $(G = (V, E), c, M)$, one may add a universal new vertex v to G, and color it with a color which does not appear in motif M. The minimum dominating set $\{v\}$ is of size 1. Vertex v cannot be part of the solution due to its color, so answering the new problem is as hard as solving the original instance. Though, this could be considered as cheating since a vertex whose color is not in M can immediately be discarded from the graph. We show that even when $\forall v \in V$, $c(v) \in M$, graphs with dominating set of size 2 can be hard to solve.

▶ **Theorem 14 (★).** Graph Motif *is* NP*-hard even for graphs with a minimum dominating set of size* 2 *and when the motif is colorful.*

4.3 Max leaf number

The *max leaf number* of a graph G, denoted $\mathrm{ml}(G)$ is the maximum number of leaves (i.e., vertices of degree 1) in a spanning tree of G. Therefore, if G is itself a tree, then $\mathrm{ml}(G)$ is simply the number of leaves of G. We will show that Graph Motif is in XP (even in W[P]) and is W[1]-hard with parameter max leaf number. In fact, we will even prove that it is W[1]-hard on trees with parameter *number of leaves in the tree plus number of distinct colors in the motif*. This strenghtens the previously known result that the problem is W[1]-hard on trees with parameter number of distinct colors in the motif [15].

▶ **Theorem 15 (★).** Graph Motif *can be solved in time* $O^*(16^k n^{10k}) = n^{O(k)}$*, where* $k = ml(G)$ *and is even in* W[P] *with respect to that parameter.*

▶ **Theorem 16 (★).** Graph Motif *is* W[1]*-hard with respect to the max leaf number plus the number of colors, even on trees.*

5 Conclusion and open problems

Figure 1 sums up the parameterized complexity landscape of Graph Motif with respect to structural parameters. For parameter maximum independent set the complexity status of Graph Motif remains unknown. Even when the problem is in FPT, polynomial kernels tend to be unlikely; be it for the natural parameter even on comb graphs or for the vertex cover number or the distance to clique. Is it the case for parameter cluster editing number?

The sparsification lemma [21] together with a straightforward reduction from 3-SAT shows that, under ETH, Graph Motif cannot be solved in time $2^{o(n)}$ on graphs with n vertices. Thus, for every parameter k bounded by n, an algorithm solving Graph Motif in $2^{o(k)}$ would disprove ETH. This is the case of four out of six parameters for which we have given an FPT algorithm; cluster editing and edge clique cover numbers are only bounded by n^2. On the one hand, it says that our algorithm running in $2^{O(k)}$ for parameter distance to clique is probably close to optimal. On the other hand, for parameter vertex cover number, for instance, we have still some room for improvement between the $2^{O(k \log k)}$-upper bound and the $2^{o(k)}$-lower bound under ETH. Can we improve the algorithm to time $2^{O(k)}$, or, on the contrary, show a stronger lower bound of $2^{o(k \log k)}$ (potentially using [26])?

References

1 Eric Alm and Adam P. Arkin. Biological Networks. *Current Opinion in Structural Biology*, 13(2):193–202, 2003.

2 Abhimanyu M. Ambalath, Radheshyam Balasundaram, Chintan Rao H., Venkata Koppula, Neeldhara Misra, Geevarghese Philip, and M. S. Ramanujan. On the Kernelization Complexity of Colorful Motifs. In *Proc. of the 5th IPEC*, volume 6478 of *LNCS*, pages 14–25. Springer, 2010.

3 Daniel Berend and Tamir Tassa. Improved bounds on bell numbers and on moments of sums of random variables. *Probab. and Math. Statist.*, 30(2):185–205, 2010.

4 Nadja Betzler, René van Bevern, Michael R. Fellows, Christian Komusiewicz, and Rolf Niedermeier. Parameterized algorithmics for finding connected motifs in biological networks. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 8(5):1296–1308, 2011.

5 Andreas Björklund, Petteri Kaski, and Lukasz Kowalik. Probably optimal graph motifs. In *Proc. of the 30th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 20 of *LIPIcs*, 2012.

6 Andreas Björklund, Petteri Kaski, Lukasz Kowalik, and Juho Lauri. Engineering motif search for large graphs. In Ulrik Brandes and David Eppstein, editors, *Proc. of the 17th ALENEX*, pages 104–118. SIAM, 2015.

7 Sebastian Böcker. A golden ratio parameterized algorithm for cluster editing. *J. Discrete Algorithms*, 16:79–89, 2012.

8 Sebastian Böcker, Florian Rasche, and Tamara Steijger. Annotating Fragmentation Patterns. In *Proc. of the 9th International Workshop Algorithms in Bioinformatics (WABI)*, volume 5724 of *LNCS*, pages 13–24. Springer, 2009.

9 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM J. Discrete Math.*, 28(1):277–305, 2014.

10 Sharon Bruckner, Falk Hüffner, Richard M. Karp, Ron Shamir, and Roded Sharan. Topology-Free Querying of Protein Interaction Networks. *Journal of Computational Biology*, 17(3):237–252, 2010.

11 Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theoretical Computer Science*, 411(40–42):3736 – 3756, 2010.

12 Marek Cygan, Marcin Pilipczuk, and Michal Pilipczuk. Known algorithms for EDGE CLIQUE COVER are probably optimal. In *Proc. of Symposium on Discrete Algorithms, SODA 2013*, pages 1044–1053. SIAM, 2013.

13 Riccardo Dondi, Guillaume Fertin, and Stéphane Vialette. Complexity issues in vertex-colored graph pattern matching. *J Discr Algo*, 9(1):82–99, 2011.

14 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013.

15 Michael R. Fellows, Guillaume Fertin, Danny Hermelin, and Stéphane Vialette. Upper and lower bounds for finding connected motifs in vertex-colored graphs. *J. Comput. Syst. Sci.*, 77(4):799–811, 2011.

16 Michael R. Fellows, Daniel Lokshtanov, Neeldhara Misra, Matthias Mnich, Frances A. Rosamond, and Saket Saurabh. The complexity ecology of parameters: An illustration using bounded max leaf number. *Theory Comput. Syst.*, 45(4):822–848, 2009.

17 Robert Ganian. Twin-cover: Beyond vertex cover in parameterized algorithmics. In *Proc. of the 6th International Symposium Parameterized and Exact Computation IPEC 2011*, volume 7112 of *LNCS*, pages 259–271. Springer, 2011.

18 Robert Ganian. Using neighborhood diversity to solve hard problems. *CoRR*, abs/1201.3091, 2012.

19 Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Data reduction and exact algorithms for clique cover. *ACM Journal of Experimental Algorithmics*, 13, 2008.

20 Sylvain Guillemot and Florian Sikora. Finding and counting vertex-colored subtrees. *Algorithmica*, 65(4):828–844, 2013.

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

22 Christian Komusiewicz and Rolf Niedermeier. New races in parameterized algorithmics. In *Proc. of Mathematical Foundations of Computer Science MFCS 2012*, volume 7464 of *LNCS*, pages 19–30. Springer, 2012.

23 Ioannis Koutis. Constrained multilinear detection for faster functional motif discovery. *Inf. Process. Lett.*, 112(22):889–892, 2012.

24 Vincent Lacroix, Cristina G. Fernandes, and Marie-France Sagot. Motif search in graphs: application to metabolic networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 3(4):360–368, 2006.

25 Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, 2012.

26 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Slightly superexponential parameterized problems. In *Proc. of SODA 2011*, pages 760–776, 2011.

27 Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41(5):960–981, 1994.

28 Rolf Niedermeier. *Invitation to Fixed Parameter Algorithms*. Lecture Series in Mathematics and Its Applications. Oxford University Press, 2006.

29 Ron Y. Pinter, Hadas Shachnai, and Meirav Zehavi. Deterministic parameterized algorithms for the graph motif problem. In *Proc. of Mathematical Foundations of Computer Science MFCS 2014*, volume 8635 of *LNCS*, pages 589–600. Springer, 2014.

30 Ron Y. Pinter and Meirav Zehavi. Algorithms for topology-free and alignment network queries. *J. Discrete Algorithms*, 27:29–53, 2014.

Kernels for Structural Parameterizations of Vertex Cover – Case of Small Degree Modulators

Diptapriyo Majumdar, Venkatesh Raman, and Saket Saurabh

The Institute of Mathematical Sciences
Chennai, India
{diptapriyom|vraman|saket}@imsc.res.in

Abstract

Vertex Cover is one of the most well studied problems in the realm of parameterized algorithms and admits a kernel with $\mathcal{O}(\ell^2)$ edges and 2ℓ vertices. Here, ℓ denotes the size of a vertex cover we are seeking for. A natural question is whether Vertex Cover admits a polynomial kernel (or a parameterized algorithm) with respect to a parameter k, that is, provably smaller than the size of the vertex cover. Jansen and Bodlaender [STACS 2011, TOCS 2013] raised this question and gave a kernel for Vertex Cover of size $\mathcal{O}(f^3)$, where f is the size of a feedback vertex set of the input graph. We continue this line of work and study Vertex Cover with respect to a parameter that is always smaller than the solution size and incomparable to the size of the feedback vertex set of the input graph. Our parameter is the number of vertices whose removal results in a graph of maximum degree two. While vertex cover with this parameterization can easily be shown to be fixed-parameter tractable (FPT), we show that it has a polynomial sized kernel.

The input to our problem consists of an undirected graph G, $S \subseteq V(G)$ such that $|S| = k$ and $G[V(G) \setminus S]$ has maximum degree at most 2 and a positive integer ℓ. Given (G, S, ℓ), in polynomial time we output an instance (G', S', ℓ') such that $|V(G')| \leq \mathcal{O}(k^5)$, $|E(G')| \leq \mathcal{O}(k^6)$ and G has a vertex cover of size at most ℓ if and only if G' has a vertex cover of size at most ℓ'. When $G[V(G) \setminus S]$ has maximum degree at most 1, we improve the known kernel bound from $\mathcal{O}(k^3)$ vertices to $\mathcal{O}(k^2)$ vertices (and $\mathcal{O}(k^3)$ edges). In general, if $G[V(G) \setminus S]$ is simply a collection of cliques of size at most d, then we transform the graph in polynomial time to an equivalent hypergraph with $\mathcal{O}(k^d)$ vertices and show that, for $d \geq 3$, a kernel with $\mathcal{O}(k^{d-\epsilon})$ vertices is unlikely to exist for any $\epsilon > 0$ unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases Parameterized Complexity, Kernelization, expansion lemma, vertex cover, structural parameterization

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.331

1 Introduction and Motivation

In the early years of parameterized complexity and algorithms, problems were almost always parameterized by the solution size. Recent research has focussed on other parameterizations based on structural parameters in the input [9], or above or below some guaranteed optimum values [13, 14, 18]. The reasons are many. Such 'non-standard' parameters are more likely to be small in practice. Also, once a problem is shown to be fixed-parameter tractable (FPT) or to have a polynomial sized kernel by a parameterization, it is natural to ask whether the problem is FPT (and admits polynomial kernel) when parameterized by a provably smaller parameter. In the same vein, if we show that a problem is W-hard under a parameterization, it is natural to ask whether it is FPT when parameterized by a provably larger parameter.

One of the earliest papers in the realm of alternate parameterization dates back to 1981. Let $\mathfrak{O}_k$ denote the set of all graphs G such that the length of the longest odd cycle is upper bounded by k. Hsu et al. [15] initiated a study of NP-hard optimization problems on $\mathfrak{O}_k$. In particular, they studied the effect of avoiding long odd cycle for the Maximum Independent Set problem and showed that a maximum sized independent set on a graph $G \in \mathfrak{O}_k$ on n vertices can be found in time $n^{\mathcal{O}(k)}$. Later, Grötschel and Nemhauser [12] did a similar study for Max-Cut and obtained an algorithm with running time $n^{O(k)}$ on a graph $G \in \mathfrak{O}_k$ on n vertices. These algorithms, using modern techniques, can be made FPT and also shown to not admit polynomial kernel unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$ [21]. Later Cai [4] did a similar study for Coloring problems. Fellows et al. [10] studied alternate parameterizatons for problems that were proven to be intractable with respect to standard parameterizations. This led to the whole new ecology program and opened up a floodgate of new and exciting research. We refer to [9] for a detailed introduction to the whole program. The kernelization results tend to be harder in this framework, as the rules need to capture the interaction of the structural parameter with the rest of the input, against simply using the property of feasible solutions in the case of the 'standard' parameterization. Vertex Cover, that asks whether a given undirected graph G has a set of size at most ℓ such that $G \setminus S$ is an independent set, for some given integer ℓ, is one of the most well studied problems in the realm of parameterized algorithms and admits an algorithm with running time $1.2738^{\ell} n^{\mathcal{O}(1)}$ and a kernel with $\mathcal{O}(\ell^2)$ edges and 2ℓ vertices [5, 19]. The set S is also called *vertex cover* of the graph. A natural question is whether Vertex Cover admits a polynomial kernel (or a parameterized algorithm) with respect to a parameter k, that is, provably smaller than the size of the vertex cover. Jansen and Bodlaender [16] first raised this question and showed that Vertex Cover admits a kernel of size $\mathcal{O}(f^3)$, where f is the size of the feedback vertex set of the input graph. Since then we have several results in this direction. For Vertex Cover parameterized by the size of the odd cycle transversal and konig vertex deletion set, there is a randomized polynomial sized kernel [17]. Vertex Cover parameterized by the deletion set to chordal graphs or perfect graphs has no polynomial kernel unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$ [2, 9]. In this paper we continue this line of work on Vertex Cover and study it with respect to a parameter that is always smaller than the solution size and incomparable to the feedback vertex set of the input graph. In particular, we consider the Vertex Cover problem parameterized by the number of vertices whose removal results in a graph of maximum degree at most x, where $x \geq 1$.

Vertex Cover parameterized by degree x Modulator (VC-x-Mod) **Parameter:** k
Input: An undirected graph G, $S \subseteq V(G)$ of size at most k such that $G[V(G) \setminus S]$ is a graph of degree at most x and an integer ℓ.
Question: Does G have a vertex cover of size at most ℓ?

Vertex Cover is known to be NP-complete even on graphs of maximum degree 3 and thus the x in VC-x-Mod must be upper bounded by 2, else we can not even hope to have an algorithm of the form $n^{f(k)}$ for any function f. On the other hand Vertex Cover is polynomial time solvable when the maximum degree is at most 2.

Let G be the input graph along with a vertex subset S such that $|S| \leq k$ and $G[V(G) \setminus S]$ has maximum degree at most 2. We call S a *degree* 2 *modulator of the graph*. By 'guessing' (i.e. trying all possible choices for) the intersection of S with the optimal vertex cover, and solving the remaining problem in polynomial time, we can find a minimum vertex cover of G in $2^k n^{\mathcal{O}(1)}$ time. This shows that VC-2-Mod is FPT. One of our main results is a polynomial kernel for VC-2-Mod.

Our Results. We obtain a kernel for VC-2-Mod with $\mathcal{O}(k^5)$ vertices, and $\mathcal{O}(k^6)$ edges. Our result is in contrast to the fact that Vertex Cover parameterized by treewidth 2 modulator (i.e. when $G[V(G) \setminus S]$ is a general graph of treewidth at most 2) has no polynomial sized kernel unless NP $\subseteq$ coNP/poly [6]. We also address the kernelization question for VC-1-Mod. Here, a kernel with $\mathcal{O}(k^3)$ vertices was already known from the result of [16] for Vertex Cover parameterized by the feedback vertex set size. This follows as the size of the feedback vertex set is at most the size of a degree 1 modulator. We improve the kernel size to $\mathcal{O}(k^2)$ vertices. More generally, we consider the Vertex Cover problem when parameterized by the size of a subset of vertices whose removal results in a graph with all components being cliques of size at most a constant d. We call a graph G *d-cluster graph* if every connected component of G is a clique and has size at most d. In particular we study the following problem:

Vertex Cover parameterized by d-CVD (VC-param-d-CVD) **Parameter:** k
Input: An undirected graph G, $S \subseteq V(G)$ of size at most k such that $G[V(G) \setminus S]$ is a d-cluster graph and an integer ℓ.
Question: Does G have a vertex cover of size at most ℓ?

Observe that VC-1-Mod and VC-param-2-CVD are the same problems. It is known that if the resulting graph is simply a clique (with no bound on the size), then a polynomial sized kernel is unlikely [2]. We show that the input graph of VC-param-d-CVD can be transformed in polynomial time to obtain an equivalent *hypergraph* with $\mathcal{O}(dk^d)$ vertices where each hyperedge is of size at most d. We also show that a kernel with $\mathcal{O}(k^{d-\epsilon})$ vertices, for any $\epsilon > 0$, is unlikely unless NP $\subseteq$ coNP/poly. We think that this idea of using hyperedges to capture certain constraints could find applications while doing a compression for the parameterized problem.

Observe that we have always assumed that the *modulator* is given as a part of the input. However, this constraint can be relaxed as both VC-2-MOD and VC-param-d-CVD admit constant factor approximation algorithms. For example, for VC-2-MOD, there is a factor 4-approximation algorithm and for VC-param-d-CVD, we can get an approximation algorithm with factor $(d+1)$(see [20] for approximation algorithms). However, we can obtain constant factor approximation algorithms can be obtained by greedily finding an obstruction (like a vertex v and any of its three neighbors in the case of VC-2-MOD) and selecting all the vertices in this obstruction to the approximate solution we are constructing. This implies that rather than demanding that modulators are given as a part of the input, we can first compute it using the polynomial time constant factor approximation algorithms and then run our kernelization algorithms using these. These will result in kernels with same asymptotic upper bounds as mentioned above.

2 Preliminaries and Definitions

By $[r]$, we mean the set $\{1, 2, \ldots r\}$. Throughout the paper we denote the *vertex cover number* (the size of a minimum vertex cover) by $vc(G)$.

▶ **Definition 1** (Kernelization)**.** Let $L \subseteq \sum^* \times \mathbb{N}$ be a parameterized language. Kernelization is a procedure that replaces the input instance (I, k) by a reduced instance (I', k') such that $k' \leq k$, $|I'| \leq g(k)$ for some function g depending only on k and $(I, k) \in L$ if and only if $(I', k') \in L$. The reduction from (I, k) to (I', k') must be computable in $poly(|I| + k)$ time.

▶ **Definition 2** (Soundness/Safeness of Reduction Rule)**.** A reduction rule that replaces an instance (I, k) of a parameterized language L by a reduced instance (I', k') is said to be sound or safe if $(I, k) \in L$ if and only if $(I', k') \in L$.

▶ **Definition 3** (Polynomial parameter transformation (PPT)). Let P_1 and P_2 are two parameterized languages. We say that P_1 is polynomial parameter reducible to P_2 if there exists a polynomial time computable function (or algorithm) $f : \sum^* \times \mathbb{N} \to \sum^* \times \mathbb{N}$, a polynomial $p : \mathbb{N} \to \mathbb{N}$ such that $(x, k) \in P_1$ if and only if $f(x, k) \in P_2$ and $k' \leq p(k)$ where $f((x, k)) = (x', k')$. We call f to be a polynomial parameter transformation from P_1 to P_2.

The following proposition gives the use of the polynomial parameter transformation for obtaining kernels for one problem from another.

▶ **Proposition 4** ([3]). *Let $P, Q \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems and assume there exists a PPT from P to Q. Furthermore, assume that the classical version of P is NP-hard and Q is in NP. Then if Q has a polynomial kernel implies that P has a polynomial kernel.*

The following powerful variation of Hall's matching theorem, known as Expansion Lemma, is used in some of our reduction rules.

▶ **Lemma 5** (q-Expansion Lemma). [11, 22, 24]) *Let q be a positive integer and G be a bipartite graph with vertex partition A and B such that $|B| > q|A|$ and there are no isolated vertices in B. Then, there exists non-empty subsets $X \subseteq A, Y \subseteq B$ obtainable in polynomial time, such that*

- *there is a q-expansion of X into Y. I.e. there is a $M \subseteq E$ such that every vertex in X is incident with exactly q edges of M. Moreover M saturates exactly $q|X|$ vertices in Y, and*
- *$N_G(Y) \subseteq X$.*

In Section 4, after applying some reduction rules, the input graph gets converted to a hypergraph where hyperedges consisting of more than 2 vertices are sometimes present in $G[S]$. We define an independent set and a vertex cover in hypergraph as follows. Recall that by $G[S]$ for a subset of vertices S (where G is a graph or a hypergraph), we denote the subgraph that consists of the vertices of S and all the (hyper) edges which are completely contained in S.

▶ **Definition 6** (Independent Set in a hypergraph). $A \subseteq V(G)$ is said to be an independent set in a hypergraph if no hyperedge is contained in $G[A]$.

▶ **Definition 7** (Vertex Cover in a hypergraph). $A \subseteq V(G)$ is said to be a vertex cover in hypergraph if for every hyperedge $e \in E(G)$, $A \cap V(e) \neq \emptyset$ where $V(e)$ be the set of vertices present in the hyperedge e.

A vertex cover in a hypergraph is also known as a *hitting set*.

3 Kernel for VC-2-Mod

Throughout this section for an input (G, S, ℓ) to VC-2-Mod we use F to denote $V(G) \setminus S$. Now, we are ready to describe the reduction rules that compress $G[F]$ to an equivalent instance whose size is polynomial in k. Note that rules will have to be applied sequentially, and after every rule is applied, we need to start from the beginning and exhaustively apply applicable rules; some of the earlier rules may become applicable after a rule is applied. We allow the input graphs to have self loops. The main reason for this is that even though input graph may not have self loops, some reduction rules (for example, Reduction Rule 8) may create self loops.

3.1 Ensuring minimum degree 3

The following reduction rules are standard for the Vertex Cover problem (see, for example, Chapter 4 of [8] for correctness of the rules).

▶ **Reduction Rule 1.** *Remove isolated vertices from G.*

▶ **Reduction Rule 2.** *If $\exists u \in V(G)$ such that there is a self loop with u, then $G' \leftarrow G \setminus \{u\}, \ell' \leftarrow \ell - 1$.*

▶ **Reduction Rule 3.** *If $\exists u \in G$ such that $deg_G(u) = 1$ and v is its unique neighbour, then $G' \leftarrow G \setminus \{u, v\}, \ell' \leftarrow \ell - 1$.*

▶ **Reduction Rule 4.** *If $\exists u \in G$ such that $deg_G(u) = 2$ and let v, w be its 2 neighbours in F, then do the followings:*

- *If $(v, w) \in E(G)$, then $G' \leftarrow G \setminus \{u, v, w\}, \ell' \leftarrow \ell - 2$*
- *If $(v, w) \notin E(G)$, then $G' \leftarrow G \setminus \{u, v, w\} \cup \{u_{new}\}, \ell' \leftarrow \ell - 1$, and make all vertices adjacent to v and w (except u) in G adjacent to u_{new}.*

When the above reduction rules are not applicable, the minimum degree in the graph is at least 3, and hence every vertex $v \in F$ has at least one neighbour in S. We partition F into F_0, F_1 and F_2 such that every connected component of $G[F_0]$ is an isolated vertex, every connected component of $G[F_1]$ is either a path (of length at least 2) or a cycle of even length (length at least 4) and every connected component of $G[F_2]$ is an odd cycle. As every component in $G[F_2]$ is an odd cycle, we interchangeably use the term component or an odd cycle to mean the same thing in $G[F_2]$. Central to the rules in this subsection is a notion of a *blocking set*, we define the notion first and then prove some properties about them.

3.2 Blocking Sets and their Properties

▶ **Definition 8** (Blocking Set and Good Set)**.** Let $B \subseteq V(G)$. We call B to be a *blocking set* if $vc(G[V(G) \setminus B]) + |B| > vc(G)$. We call a *blocking set* B to be a *minimal blocking set* if no proper subset of B is a blocking set. A set $B \subseteq F$ is called a *good set* if it is not a *blocking set*.

If an algorithm picks the vertices of a blocking set B into a solution, then any way to complete it to an optimum vertex cover results in a non-optimal solution. So the blocking set 'blocks' the completion step from resulting into an optimal solution. For example, in a cycle of even length, the end points of any edge form a blocking set as no optimum solution for the cycle contains two vertices of the same edge. We will apply these notions to $G[F]$.

For two vertices a_i, a_j of a cycle C where vertices are ordered as $a_0, a_1, \ldots, a_{|C|-1}$ with $i < j \mod |C|$, by $dist(a_i, a_j)$ we mean the length (the number of edges) in the *clockwise* path that goes through the vertices $a_{i+1}, a_{i+2}, \ldots, a_{j-1}$ where all the subscripts are taken mod $|C|$. The following statement is easy to verify.

▶ **Observation 9.** *In a cycle, no single vertex forms a blocking set; hence minimal blocking sets are of size at least 2 in cycles.*

▶ **Lemma 10** ($\star$[1])**.** *Let $B \subseteq V(F)$ be a minimal blocking set in $G[F]$. Then there exists a unique C for which $B \subseteq V(C)$ where C is a component of $G[F]$.*

[1] Due to lack of space, the proofs of results marked $\star$, and the proof of correctness and the polynomial runtime of our reduction rules will appear in the full version.

▶ **Theorem 11** (⋆).
- *In an odd cycle, the only minimal blocking sets are of size* 3 *where the clockwise distance between every pair of them is odd.*
- *In an even cycle, the only minimal blocking sets are of size* 2 *where the clockwise distance between every pair of them is odd.*

▶ **Definition 12** (Bad Component and Nice Component)**.** Let C be a cycle in $G[F_2]$. If there exists an independent set $A \subseteq S$ of size at most 3 such that $N_G(A) \cap C$ contains a blocking set, then we call C a *bad component*. A component is said to be a *nice component* if it is not a bad component.

We partition the set of bad components (in F_2) as $\mathcal{B}_1, \mathcal{B}_2$ and $\mathcal{B}_3$ where
$\mathcal{B}_1 = \{C | C$ is a component and $\exists x \in S$ such that $N_G(x) \cap C$ contains a blocking set$\}$.
$\mathcal{B}_2 = \{C | C$ is a component and $\exists x, y \in S, (x, y) \notin E(G)$ such that $C \cap (N_G(x) \cup N_G(y))$ contains a blocking set$\} \setminus \mathcal{B}_1$.
$\mathcal{B}_3 = \{C | C$ is a component and $\exists x, y, z \in S, \{x, y, z\}$ is independent set such that $C \cap (N_G(x) \cup N_G(y) \cup N_G(z))$ contains a blocking set$\} \setminus (\mathcal{B}_1 \cup \mathcal{B}_2)$.

By $\mathcal{B}_4$ we denote the set of nice components. The following observation follows from the definitions.

▶ **Observation 13.** *If $C \in \mathcal{B}_2$ then for any blocking set B in C, $B \not\subset N_G(x)$ for any $x \in S$ and if $C \in \mathcal{B}_3$, then for any blocking set B in C, $B \not\subset N_G(A)$ for any independent set of size at most* 2 *in S.*

3.3 Towards bounding the number of components

In this subsection we describe three rules, two of which are powerful to help us bound the number of components in $G[F]$.

▶ **Reduction Rule 5** (NiceComponent Rule)**.** *Let C be a nice component in F. Then $G' \leftarrow G \setminus C, \ell' \leftarrow \ell - vc(G[C])$.*

▶ **Reduction Rule 6.** *If there exists a vertex $x \in S$ such that $vc(G[F \setminus N_G(x)]) + |N_G(x) \cap F| \geq vc(G[F]) + |S| + 1$, then $G' \leftarrow G \setminus \{x\}, \ell' \leftarrow \ell - 1$.*

▶ **Reduction Rule 7.** *If there exists $x, y \in S, (x, y) \notin E(G)$ such that $vc(G[F \setminus N_G(\{x, y\})]) + |N_G(\{x, y\}) \cap F| \geq vc(G[F]) + |S| + 1$, then add edge (x, y) into G.*

Note that while this rule does not decrease the size of the graph or ℓ, it does enable the applicability of some rules (for example Reduction Rule 8 which is stated later).

▶ **Lemma 14.** *If Reduction Rules 6 and 7 are not applicable, then the following statements are true. Let M be a maximum matching of $G[F_1]$, and let M_1 be a maximum matching of $G[F_2]$.*

1. *Then, for every $x \in S$,*
   - *$|N_G(x) \cap F_0| \leq |S|$.*
   - *$N_G(x) \cap F$ contains both end points of at most $|S|$ edges of M.*
   - *$N_G(x) \cap F$ contains both end points of at most $|S| + c$ edges of M_1 where c is the number of odd cycles in $G[F_2]$.*
   - *$N_G(x) \cap F$ contains a blocking set in at most $|S|$ cycles in $\mathcal{B}_1$.*
2. *For every pair x, y of vertices in S such that $(x, y) \notin E(G)$,*
   - *$N_G(\{x, y\}) \cap F$ contains both end points of at most $|S|$ edges of M.*

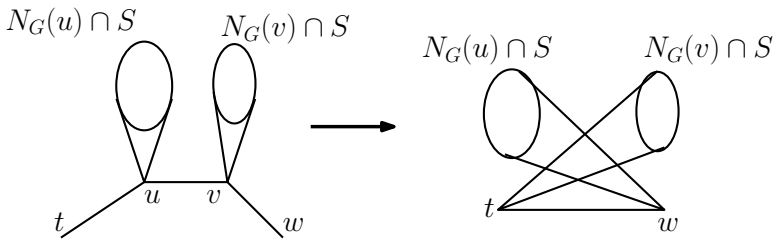


Figure 1 An Illustration of Reduction Rule 8.

- *$N_G(\{x, y\}) \cap F$ contains both end points of at most $|S| + c$ edges of M_1 where c is the number of odd cycles in $G[F_2]$.*
- *$N_G(\{x, y\}) \cap F$ contains blocking set in at most $|S|$ cycles in $\mathcal{B}_2$.*

Proof sketch. The set F_0 contains isolated vertices. Therefore, if a vertex $x \in S$ is adjacent to at least $|S| + 1$ vertices in F_0, then any vertex cover C such that $x \notin C$ must pick $vc(F)$ vertices from F (which do not contain any vertex from F_0) and at least $|S| + 1$ vertices from F_0. Therefore, Reduction Rule 6 becomes applicable. Hence, $|N_G(x) \cap F_0| \leq |S|$. By similar arguments and using properties of blocking sets in odd cycles, we can prove the other facts when Reduction Rule 6 is not applicable.

The graph $G[F_1]$ is bipartite and let $vc(G[F_1]) = |M|$. Now, any vertex cover C such that $x, y \notin C$ with $(x, y) \notin E(G), x, y \in S$ must contain $vc(F)$ vertices from F containing exactly one endpoint from every edge of M and at least $|S| + 1$ other vertices from those edges of M both of whose end points are present in $N_G(\{x, y\})$. Therefore, when reduction rule 7 is not applicable, then $N_G(\{x, y\})$ must contain both end points of at most $|S|$ edges in M. By using similar arguments and propertes of blocking sets in odd cycles, we can prove the other facts when reduction rule 7 is not applicable. ◀

▶ **Corollary 15.** *When Reduction Rules 6, 7 are not applicable, then*

- $|\mathcal{B}_2| \leq k\binom{k}{2}$.
- $|\mathcal{B}_1| \leq k^2$.
- $|F_0| \leq k^2$.

Proof sketch. We know that for every odd cycle $C \in \mathcal{B}_1$, there exists a vertex $x \in S$ such that $N_G(x)$ contains a blocking set in C. By Lemma 14, for every $x \in S$, there are at most k cycles in $\mathcal{B}_1$ such that $N_G(x)$ contains a blocking set in each of those cycles. Therefore, $|\mathcal{B}_1| \leq k^2$. By using a similar argument, we can justify the other claims.

Another easy consequence of Lemma 14 is that $|F_0| \leq k^2$. ◀

By a similar argument, we can bound $|\mathcal{B}_3|$ as well, but we take this up in Section 3.5 by a different argument which gives a better bound.

3.4 Bounding the number of vertices in F_1

Now we describe a rule that helps bound the number of vertices in F_1. Here instead of bounding the number of components in F_1 and the number of vertices in each component, we directly bound the number of edges in the maximum matching M of $G[F_1]$. The rule is a slight variation of one proposed by Jansen and Bodlaender [16] for Vertex Cover parameterized by the feedback vertex set number.

▶ **Reduction Rule 8** (Edge Rule). *Let $\exists(u,v) \in E(G[F])$ such that $(N_G(u) \cap S) \cap (N_G(v) \cap S) = \emptyset$ and $\forall x \in N_G(u) \cap S, \forall y \in N_G(v) \cap S : (x,y) \in E(G)$ (i.e. $N_G(u) \cap S$ and $N_G(v) \cap S$ induce a complete bipartite graph). (See Figure 1 for an illustration.) Then do the following.*

- *Delete u, v from the graph.*
- *If u has a neighbour t in F which is not v, then make t adjacent to every vertex in $N_G(v) \cap S$.*
- *If v has a neighbour w in F which is not u, then make w adjacent to every vertex in $N_G(u) \cap S$.*
- *If the vertices t, w exist, then they are unique and add the edge (t,w).*
- *Set ℓ' to $\ell - 1$.*

Then we have the following lemma.

▶ **Lemma 16** ($\star$). *When reduction rules 1 to 8 are not applicable $G[F_1 \cup F_0]$ has $\mathcal{O}(k^3)$ vertices.*

3.5 Bounding the the number of odd cycles

We use the Expansion Lemma 5 to get an upper bound on $\mathcal{B}_3$. We construct a bipartite graph as $H = (S_3, \mathcal{B}_3, E)$ where S_3 is the set of all independent sets of size 3 from S, and $E(H)$ is defined as follows. $E(H) = \{(I,L) | \exists B \subseteq V(L)$ such that B is a blocking set of size 3 and $B \subseteq N_G(I)\}$.

▶ **Reduction Rule 9.** *If $|\mathcal{B}_3| > 5|S_3|$, then apply Expansion lemma 5 with $q = 5$ from S_3 to $\mathcal{B}_3$ to get $A \subseteq S_3, B \subseteq \mathcal{B}_3$ such that $N_H(B) \subseteq A$ and there is a 5-expansion from A to B.*

Associated with every $(x,y,z) \in A$, there are 5 distinct cycles in B. Pick one of those 5 cycles for each such $\{x,y,z\} \in A$. Let $C_{p_1}, \ldots, C_{p_{|A|}}$ be collection of such cycles. Then set $G' \leftarrow G \setminus (C_{p_1} \cup \ldots \cup C_{p_{|A|}}), \ell' \leftarrow \ell - (\sum_{i=1}^{|A|} vc(C_{p_i}))$.

Now it is easy to show that

▶ **Corollary 17.** *If Reduction Rule 9 is not applicable, then $|\mathcal{B}_3| \leq 5\binom{k}{3}$.*

Finally the following lemma follows from Reduction Rule 5 (as if this rule is not applicable, every component is bad) and Corollaries 15 and 17.

▶ **Lemma 18.** *If none of the above reduction nules is applicable, then the number of components in $G[F_2]$ is $\mathcal{O}(k^3)$.*

3.6 Bounding $G[F_2]$ and Putting things together

▶ **Lemma 19.** *When Reduction Rules 1 to 9 are not applicable, the number of vertices in $G[F_2]$ is $\mathcal{O}(k^5)$.*

Proof. By Reduction Rule 8, for every edge $(u,v) \in E(G[F])$, there is either a vertex $x \in S$ such that $x \in N(u) \cap N(v)$ or there exists a pair of non-adjacent vertices $x, y \in S$ such that $x \in N(u), y \in N(v)$. In the former case, we associate the vertex x with the edge (u,v) and to the pair of vertices (x,y) in the case of the latter. By Lemma 14 every $x \in S$ is adjacent to both end points of at most $k + c$ edges of M_1 in C, and every pair of non-adjacent vertices $x, y \in S$, are together adjacent to both end points of at most $k + c$ edges of M_1. Therefore, $|M_1| \leq (k + \binom{k}{2})(k + c) = \mathcal{O}(k^5)$ as c is $\mathcal{O}(k^3)$. It follows that $|V(G[F_2])| = 2|M_1| + c = \mathcal{O}(k^5)$. ◀

The following theorem is an immediate consequence of Lemma 19 and Lemma 16. The bound on the number of edges follows because the number of edges in $G[S]$ is $\mathcal{O}(k^2)$, the number of edges in $G[F]$ is $\mathcal{O}(k^5)$ (as each vertex has degree at most 2 in $G[F]$, and the number of edges between F and S can be at most $\mathcal{O}(k^6)$.

▶ **Theorem 20.** VC-2-Mod *has a kernel consisting of* $\mathcal{O}(k^5)$ *vertices and* $\mathcal{O}(k^6)$ *edges.*

4 Vertex Cover parameterized by bounded cluster vertex deletion set

Now we consider the Vertex Cover problem when parameterized by the size of the degree 1 modulator. Here the resulting graph after removal of the modulator is a collection of isolated vertices and edges – i.e. a collection of cliques of size at most 2. In fact, we will consider the general problem VC-param-d-CVD.

If there is no bound on the sizes of the cliques in $G[F]$, then it is known that the problem has no polynomial kernel unless NP $\subseteq$ coNP/poly. This follows from a result of Bodlaender et al. [2] who showed this infeasibility of polynomial kernel for Vertex Cover when parameterized by *clique deletion set* which is a set of vertices whose removal results in a clique.

We start with a definition, similar to the notion of *Bad and Nice component* in the kernelization of VC-2-Mod.

▶ **Definition 21** (Bad Clique and Nice Clique)**.** A clique C of $G[F]$ is said to be a *bad clique* if $\exists A \subseteq S$ such that A is independent and $|A| \leq d$ and $V(C) \subseteq N_G(A)$. A clique is said to be a *nice clique* if it is not a *bad clique*.

Observe that any clique C in $G[F]$, that contains a vertex which has no neighbour in S, is a nice clique. Now we proceed to state the list of reduction rules for this problem. As a preprocessing, we only require that isolated vertices are removed (i.e. there is no need to even make the graph minimum degree 3 using the rules of Section 3.1).

▶ **Reduction Rule 10.** *For every nice clique* C *of* $G[F]$*, delete it to obtain* $G' \leftarrow G \setminus V(C)$ *and make* $\ell' \leftarrow \ell - (|V(C)| - 1)$.

Note that when Reduction rule 10 is not applicable, every vertex in F has a neighbour in S.

▶ **Reduction Rule 11.** *Let* $\exists I \subseteq S$ *such that* $|I| \leq d-1, I$ *is an independent set and* $N_G(I)$ *contains all vertices of at least* $|S|+1$ *cliques in* F*, then do the following:*

- *If* $|I| = 1$*, then* $G' \leftarrow G \setminus I, \ell' \leftarrow \ell - 1$.
- *If* $2 \leq |I| \leq (d-1)$*, then add the hyperedge* $\{x_1, \ldots, x_{d-1}\}$ *into* G *where* $I = \{x_1, \ldots, x_{d-1}\}$.

We partition the set of bad cliques in $G[F]$ into 2 parts as follows.
$Z_1 = \{Z | Z$ such that $\exists I \subseteq S, |I| \leq d-1, V(Z) \subseteq N_G(I)\}$.
$Z_2 = \{Z | Z$ such that $\exists I \subseteq S, |I| = d, V(Z) \subseteq N_G(I)\} \setminus Z_1$.
Note that hyperedges consisting of more than 2 vertices are present only in S. Therefore, for any vertex $x \in S$, $N_G(x) \cap F = \{y \in F | (x,y) \in E(G)\}$ and for any vertex $u \in F$, $N_G(u) \cap S = \{v \in S | (u,v) \in E(G)\}$. For every vertex x, let $HE(x) = \{e \in E(G) | x \in V(e)\}$. For every hyperedge e let $V(e)$ be the set of vertices that are present in the hyperedge e. The following two rules are due to [1].

▶ **Reduction Rule 12** (Vertex and Edge Domination Rule)**.** *Let there be two vertices* x, y *such that* $HE(x) \subseteq HE(y)$*, then delete* x *from* G*. Similarly, if there are two hyperedges* e_1, e_2 *such that* $V(e_1) \subseteq V(e_2)$*, then delete* e_2 *from* G.

▶ **Corollary 22.** *When none of the above reduction rules are applicable,* $|Z_1| \leq k(\sum_{i=1}^{d-1} \binom{k}{i})$.

Proof. By the definition of Z_1, for every clique $C \in Z_1$, there exists an independent set X of size at most $d-1$ of S such that $N_G(X) \subseteq C$. By Reduction Rule 11, for every independent set X of size at most $d-1$, $N_G(X)$ contains all end points of at most k cliques. Therfore, the number of cliques in Z_1 is at most $k(\sum_{i=1}^{d-1} \binom{k}{i})$. ◀

Using similar arguments, we can give a bound for $|Z_2|$ of $\mathcal{O}(k^{d+1})$, but we give an improved bound using the expansion lemma. Let Z be a clique in Z_2 consisting of d vertices. For every $u, v \in V(Z)$, we have that $N_G(u) \cap N_G(v) \cap S = \emptyset$ by definition. But for every $u \in V(Z)$, we have $N_G(u) \cap S \neq \emptyset$. Now, we construct a bipartite graph $H(S_B, Z_2, J)$. Let $S_B = \{X \subseteq S | X$ is an independent set in G and $|X| = d\}$. We add an edge (I, Z) in J if $V(Z) \subseteq N_G(I)$.

▶ **Reduction Rule 13.** *If* $|Z_2| > (d+1)|S_B|$*, then apply Expansion Lemma 5 with* $q = (d+1)$ *from* S_B *to* Z_2 *to obtain* $P_B \subseteq S_B, Q_B \subseteq Z_2$ *such that* $N_H(Q_B) \subseteq P_B$*. For every* $X \in S_B$*, add the hyperedges* $\{x_1, \ldots, x_d\}$ *where* $X = \{x_1, \ldots, x_d\}$*.*

▶ **Theorem 23.** VC-PARAM-d-CVD *has a compression of size* $\mathcal{O}(k^d)$*. In other words, when no reduction rule is applicable, the resulting hypergraph has* $\mathcal{O}(k^d)$ *vertices. Each hyperedge is of size at most* d*, and hyperedges of size more than* 2 *are present only in* S*.*

Proof. By Reduction Rule 10, every clique $Z \in G[F]$ is a bad clique. Every bad clique is of 2 types. By Corollary 22, $|Z_1| \leq k(\sum_{i=1}^{d-1} \binom{k}{i})$. Now for every clique Z of Z_2, there exists an independent set $I \subseteq S$ of size d such that $N_G(I) \subseteq Z$. There are at most $\binom{k}{d}$ independent sets of size d in S. As Reduction Rule 13 is not applicable, $|Z_2| \leq (d+1)\binom{k}{d}$. Therefore, the resulting hypergraph has a kernel of size at most $k(\sum_{i=1}^{d-1} \binom{k}{i}) + (d+1)\binom{k}{d} = \mathcal{O}(k^d)$. ◀

When $G[F]$ is a graph of degree at most 1, every component of $G[F]$ is either an isolated vertex or an edge. Setting $d = 2$ in the above theorem, we get the following. The edge bound follows as every vertex in F has degree at most 1 within F and at most k into S. Note also that the resulting hypergraph is simply a graph.

▶ **Corollary 24.** VC-1-MOD *has a kernel on* $\mathcal{O}(k^2)$ *vertices and* $\mathcal{O}(k^3)$ *edges.*

5 Lower Bounds

Now, we prove a lower bound, under complexity theoretic assumptions for the size of the kernel of the problems we considered in this paper. We prove this by giving a polynomial parameter transformation (see definition in Section 2) from d-CNF-SAT to our problem(s) and use the following theorem due to Dell and Melkebeek [7]. Here d-CNF-SAT is the problem of testing the satisfiability of a d-CNF formula, a boolean formula where the clauses are in CNF form with at most d variables each.

▶ **Theorem 25** (Lower Bound for d-CNF-SAT)**.** d-CNF-SAT *parameterized by* n*, the number of variables, has no kernel of size* $\mathcal{O}(n^{d-\epsilon})$ *for any* $d \geq 3, \epsilon > 0$ *unless* NP $\subseteq$ coNP/poly.

There is a standard reduction (see for example [23], when $d = 3$) from d-CNF-SAT to VC-PARAM-d-CVD.

▶ **Theorem 26** (⋆). *There exists a polynomial parameter transformation from the* d-CNF-SAT *parameterized by the number of variables to* VC-PARAM-d-CVD. *In* VC-PARAM-d-CVD, *the size of the modulator is twice the number of variables in the* d-CNF-SAT *formula.*

The following theorem follows from Theorem 26 and Proposition 4.

▶ **Theorem 27.** VC-PARAM-d-CVD *has no kernel of size* $\mathcal{O}(k^{d-\epsilon})$ *for any* $d \geq 3, \epsilon > 0$ *unless* NP $\subseteq$ coNP/poly.

As a collection of cliques of size at most 3 is a subclass of graphs with degree at most 2, we have the following corollary.

▶ **Corollary 28.** VC-2-MOD *has no kernel of size* $\mathcal{O}(k^{3-\epsilon})$ *unless* NP $\subseteq$ coNP/poly.

6 Conclusion

In this paper we gave a polynomial kernel for VC-2-MOD. There is a gap between upper and lower bounds on the kernel sizes we obtained; it would be interesting to bridge this gap. It is known that VERTEX COVER admits a randomized polynomial kernel parameterized by the odd cycle transversal number of the graph (minimum number of vertices whose deletion results in a bipartite graph). Is it possible to obtain a deterministic kernel for this parameterization (maybe using some ideas from this paper)? We think this might be easier and probably the first step towards obtaining a deterministic kernel for the ODD CYCLE TRANSVERSAL problem itself.

Acknowledgements. We thank the referees whose comments helped unify some of the reduction rules and improved the presentation of the paper.

References

1 Faisal N. Abu-Khzam. A kernelization algorithm for d-hitting set. *J. Comput. Syst. Sci.*, 76(7):524–531, 2010.
2 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM J. Discrete Math.*, 28(1):277–305, 2014.
3 Hans L. Bodlaender, Stéphan Thomassé, and Anders Yeo. Kernel bounds for disjoint cycles and disjoint paths. *Theor. Comput. Sci.*, 412(35):4570–4578, 2011.
4 Leizhen Cai. Parameterized complexity of vertex colouring. *Discrete Applied Mathematics*, 127(3):415–429, 2003.
5 Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theor. Comput. Sci.*, 411(40-42):3736–3756, 2010.
6 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. On the hardness of losing width. *Theory Comput. Syst.*, 54(1):73–82, 2014.
7 Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. *J. ACM*, 61(4):23:1–23:27, 2014.
8 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity.* Texts in Computer Science. Springer, 2013.
9 Michael R. Fellows, Bart M. P. Jansen, and Frances A. Rosamond. Towards fully multivariate algorithmics: Parameter ecology and the deconstruction of computational complexity. *Eur. J. Comb.*, 34(3):541–566, 2013.
10 Michael R. Fellows, Daniel Lokshtanov, Neeldhara Misra, Matthias Mnich, Frances A. Rosamond, and Saket Saurabh. The complexity ecology of parameters: An illustration using bounded max leaf number. *Theory Comput. Syst.*, 45(4):822–848, 2009.

11 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, Geevarghese Philip, and Saket Saurabh. Hitting forbidden minors: Approximation and kernelization. In *STACS*, pages 189–200, 2011.

12 Martin Grötschel and George L. Nemhauser. A polynomial algorithm for the max-cut problem on graphs without long odd cycles. *Math. Programming*, 29(1):28–40, 1984.

13 Gregory Gutin, Eun Jung Kim, Stefan Szeider, and Anders Yeo. A probabilistic approach to problems parameterized above or below tight bounds. *J. Comput. Syst. Sci.*, 77(2):422–429, 2011.

14 Gregory Gutin and Anders Yeo. Constraint satisfaction problems parameterized above or below tight bounds: A survey. In *The Multivariate Algorithmic Revolution and Beyond - Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*, volume 7370 of *Lecture Notes in Computer Science*, pages 257–286. Springer, 2012.

15 Wen Lian Hsu, Yoshiro Ikura, and George L. Nemhauser. A polynomial algorithm for maximum weighted vertex packings on graphs without long odd cycles. *Math. Programming*, 20(2):225–232, 1981.

16 Bart M. P. Jansen and Hans L. Bodlaender. Vertex cover kernelization revisited - upper and lower bounds for a refined parameter. *Theory Comput. Syst.*, 53(2):263–299, 2013.

17 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. In *FOCS*, pages 450–459, 2012.

18 Daniel Lokshtanov, N. S. Narayanaswamy, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. Faster parameterized algorithms using linear programming. *ACM Transactions on Algorithms*, 11(2):15:1–15:31, 2014.

19 George L. Nemhauser and Leslie E. Trotter Jr. Vertex packings: Structural properties and algorithms. *Math. Program.*, 8(1):232–248, 1975.

20 Michael Okun and Amnon Barak. A new approach for approximating node deletion problems. *Inf. Process. Lett.*, 88(5):231–236, 2003.

21 Fahad Panolan and Ashutosh Rai. On the kernelization complexity of problems on graphs without long odd cycles. In *COCOON 2012*, volume 7434 of *LNCS*, pages 445–457. Springer, 2012.

22 Elena Prieto. *Systematic kernelization in FPT algorithm design*. PhD thesis, The University of Newcastle, Australia, 2005.

23 Michael Sipser. *Introduction to the theory of computation*. PWS Publishing Company, 1997.

24 Stéphan Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM Transactions on Algorithms*, 6(2), 2010.

Parameterized Complexity of Critical Node Cuts

Danny Hermelin[1], Moshe Kaspi[1], Christian Komusiewicz[2], and Barak Navon[1]

1 **Department of Industrial Engineering and Management, Ben-Gurion University, Israel**
hermelin@bgu.ac.il, moshe@exchange.bgu.ac.il, baraknav@post.bgu.ac.il

2 **Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany**
christian.komusiewicz@tu-berlin.de

Abstract

We consider the following graph cut problem called Critical Node Cut (CNC): Given a graph G on n vertices, and two positive integers k and x, determine whether G has a set of k vertices whose removal leaves G with at most x connected pairs of vertices. We analyze this problem in the framework of parameterized complexity. That is, we are interested in whether or not this problem is solvable in $f(\kappa) \cdot n^{O(1)}$ time (*i.e.*, whether or not it is *fixed-parameter tractable*), for various natural parameters κ. We consider four such parameters:

- The size k of the required cut.
- The upper bound x on the number of remaining connected pairs.
- The lower bound y on the number of connected pairs to be removed.
- The treewidth w of G.

We determine whether or not CNC is fixed-parameter tractable for each of these parameters. We determine this also for all possible aggregations of these four parameters, apart from $w + k$. Moreover, we also determine whether or not CNC admits a polynomial kernel for all these parameterizations. That is, whether or not there is an algorithm that reduces each instance of CNC in polynomial time to an equivalent instance of size $\kappa^{O(1)}$, where κ is the given parameter.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.1 Combinatorics, G.2.2 Graph Theory

Keywords and phrases graph cut problem, NP-hard problem, treewidth

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.343

1 Introduction

In 2013 a polio virus struck Israel. The virus spread in alarming speed, creating a nationwide panic of parents concerned about the well-being of their children. It was obvious to the Israeli health department that vaccinating all Israeli children is not a practical solution in the given time frame. Thus it became clear that some areas of the country should be vaccinated first in order to stop the spread of the virus as quickly as possible. Let us represent a geographic area as a vertex of a graph, and the roads between areas as edges of the graph. In this setting, vaccinating an area corresponds to deleting a certain vertex from the graph. Thus, the objective of stopping the virus from spreading translates to minimizing the number of *connected pairs* (two vertices which are in the same connected component) in the corresponding graph after applying the vaccination.

This scenario can be modeled by the following graph-theoretic problem called Critical Node Cut (CNC). In this problem, we are given an undirected simple graph G and two integers k and x. The objective is to determine whether there exists a set $C \subseteq V(G)$ of

10th International Symposium on Parameterized and Exact Computation (IPEC 2015).
Editors: Thore Husfeldt and Iyad Kanj; pp. 343–354

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

at most k vertices in G, such that the graph $G - C$ which results from removing C from G, contains at most x connected pairs. In this sense, the cut C is considered *critical* since removing it from G leaves few (at most x) connected pairs. For convenience, throughout the paper we will count *ordered* connected pairs; *i.e.*, pairs $(u, v) \in V(G) \times V(G)$, $u \neq v$, where u and v belong to same connected component in $G - C$.

The goal of CNC is thus, roughly speaking, to destroy the connectivity of a given graph as much as possible given a certain budget for deleting vertices. From this point of view, CNC fits nicely to the broad family of *graph-cut problems*. Graph-cut problems have been studied widely and are among the most fundamental problems in algorithmic research. Examples include Min Cut, Max Cut, Multicut, Multiway Cut, Feedback Vertex Set, and Vertex Cover (see *e.g.* [19] for definitions of these problems). The latter is the special case of CNC with $x = 0$. Since Vertex Cover is one of the most important problems in the theory of algorithmic design for NP-hard problems, CNC provides a natural test bed to see which of the techniques from this theory can be extended, and to what extent.

Previous Work and Applications. The CNC problem has been studied from various angles. The problem was shown to be NP-complete in [3] (although its NP-completeness follows directly from the much earlier NP-completeness result for Vertex Cover). In trees, a weighted version of CNC is NP-complete whereas the unweighted version can be solved in polynomial time [12]. The case of bounded treewidth can be solved using dynamic programming in $O(n^{w+1})$ time, where n is the number of vertices in the graph and w is its treewidth [1]. Local search [3] and simulated annealing [28] were proposed as heuristic algorithms for CNC. Finally, in [29] an approximation algorithm based on randomized rounding was developed.

Due to its generic nature, the CNC problem has been considered in various applications. One example application is the virus vaccination problem discussed above [3]. Other interesting applications include protecting a computer/communication network from corrupted nodes, analyzing anti-terrorism networks [23], measuring centrality in brain networks [21], insulin signaling [27], and protein-protein interaction network analysis [6].

Our Results. From reviewing the literature mentioned above, it is noticeable that an analysis of CNC from the perspective of parameterized complexity [13] is lacking. The purpose of this paper is to remedy this situation. We examine CNC with respect to four natural parameters along with all their possible combined aggregations. The four basic parameters we examine are:

- The size k of the solution (*i.e.*, the critical node cut) C.
- The bound x on the number of connected pairs in the resulting graph $G - C$.
- The number of connected pairs y to be removed from G; if G is connected and has n vertices then $y = n(n-1) - x$.
- The treewidth w of G.

Table 1 summarizes all we know regarding the complexity of CNC with respect to these four parameters and their aggregation.

Let us briefly go through some of the trivial results given in the table above. First note that CNC with $x = 0$ is precisely the Vertex Cover problem, which means that CNC is not in FPT (and therefore has no polynomial kernel) for parameter x unless P=NP. This also implies that the problem is unlikely to admit a polynomial kernel even when parameterized by $w+x$, since such a kernel would imply a polynomial kernel for Vertex Cover parameterized by the treewidth w which is known to cause the collapse of the polynomial hierarchy [5, 15].

Table 1 Summary of the complexity results for Critical Node Cut.

Parameter				Result	
k	x	y	w	FPT	P-Kernel
✓				NO (Thm. 1)	NO (Thm. 1)
	✓			NO	NO
		✓		YES (Thm. 13)	NO (Thm. 14)
			✓	NO (Thm. 5)	NO (Thm. 14)
✓	✓			YES (Thm. 3)	YES (Thm. 4)
✓		✓		YES (Thm. 13)	NO (Thm. 14)
✓			✓	**?**	NO (Thm. 14)
	✓	✓		YES	YES
	✓		✓	YES (Thm. 12)	NO
		✓	✓	YES (Thm. 13)	NO (Thm. 14)
✓	✓	✓		YES	YES
✓	✓		✓	YES (Thm. 3)	YES (Thm. 4)
✓		✓	✓	YES (Thm. 13)	NO (Thm. 14)
	✓	✓	✓	YES	YES
✓	✓	✓	✓	YES	YES

Next, notice that if our input graph G has no isolated vertices, we have $x + y = \Omega(n)$, and therefore CNC is FPT and has a polynomial kernel for $x + y$ (as isolated vertices can safely be discarded). This of course means that the same applies for parameters $k+x+y$, $x+y+w$, and $k + x + y + w$.

Our first result, stated in Theorem 1, shows that CNC parameterized by k is W[1]-hard. Thus, CNC is unlikely to have an FPT algorithm under this parameterization. We then show in Theorem 3 and Theorem 4, that when considering $x + k$ as a parameter, we can extend two classical Vertex Cover techniques to the CNC problem. Our main technical result is stated in Theorem 5, where we prove that CNC is W[1]-hard with respect to w, the treewidth of the input graph. This is somewhat surprising since not many graph cut problems are known to be W[1]-hard when parameterized by treewidth. Also, the result complements nicely the $O(n^{w+1})$-time algorithm of [1] by showing that this algorithm cannot be improved substantially. We complement this algorithm from the other direction by showing in Theorem 12 that CNC can be solved in $f(w + x) \cdot n^{O(1)}$ time. Finally, we show in Theorem 13 and Theorem 14 that CNC is FPT with respect to y, and has no polynomial kernel even for $y + w + k$. Due to lack of space, most proofs are deferred to a full version of the article.[1]

Related Work. This paper belongs to a recent extensively explored line of research in parameterized complexity where various types of graph cut problems are analyzed according to various natural problem parameterizations. This line of research can perhaps be traced back to the seminal paper of Marx [24] who studied five such problems, and in the process introduced the fundamental notion of *important separators*. This paper paved the way to several parameterized results for various graph cut problems, including Multicut [7, 20, 22, 24, 25, 26, 30], MultiwayCut [9, 10, 11, 20, 24, 30], and Steiner Multicut [8]. A particularly closely related problem to CNC is the so-called Vertex Integrity problem where we want to remove k vertices from a graph such that the largest connected component

[1] A preliminary full version can be obtained at `http://arxiv.org/abs/1503.06321`.

in the remaining graph has a bounded number of vertices. Fellows and Stueckle [18] were the first to analyze this problem from a parameterized point of view; we refer the reader to [14] for a detailed overview. The edge deletion variant of Vertex Integrity has also been studied [16].

2 Parameters k and $k + x$

We now consider the parameters k and $k + x$ for CNC. We first show that the problem is W[1]-hard for k. To this end, we devise a reduction from Clique. From an instance (G, ℓ) of Clique, which asks whether G contains a complete subgraph of order ℓ, we construct H, the graph of our CNC instance, as follows: Replace each edge in G by n parallel edges, and then subdivide each of the new edges once. Next, add an edge in H between each pair of nonadjacent vertices of G. Finally, set $k := \ell$.

▶ **Theorem 1.** Critical Node Cut *is* W[1]*-hard with respect to* k*.*

We next show that the above result holds also for some restricted subclasses. A *split graph* is a graph in which the vertices can be partitioned into a clique and an independent set. We slightly modify the construction by adding all the edges missing between every pair of non-dummy vertices. In this way, the vertices of G form a clique and the dummy vertices form an independent set, while all arguments in the proof above still hold. For a fixed integer $d \geq 1$, a graph is called *d-degenerate* if each of its subgraphs has a vertex with a degree of at most d. For $d = 1$ (*i.e.*, a forest), the CNC problem has a polynomial algorithm based on dynamic programming [12]. We modify the construction in the proof above by subdividing all the edges except those that are adjacent to dummy vertices. This results in a 2-degenerate graph, and also a bipartite graph with one side containing all vertices of G and the other containing all the dummy vertices. By a slightly more careful (yet still along the same lines) argument it can be shown that the conclusion of Theorem 1 still stands.

▶ **Corollary 2.** Critical Node Cut *remains* W[1]*-hard with respect to* k *even if the input graph is split, bipartite, or* d*-degenerate for any fixed* $d \geq 2$*.*

We next consider the parameter $k + x$. We will show that the basic techniques known for the case of $x = 0$, *i.e.*, Vertex Cover, can be extended to the case where $x > 0$. First, a simple branching strategy can be developed into an FPT algorithm for the parameter $k + x$.

▶ **Theorem 3.** Critical Node Cut *is* FPT *with respect to* $k + x$*.*

The running time can be improved by using a more elaborate approach in the last step. For example, isolated edges can be dealt with in a dynamic programming subroutine. Then the remaining instance on which brute-force has to be applied has at most $1.5x$ vertices. Next, we show that a simple "high-degree rule" leads to a polynomial kernel.

▶ **Theorem 4.** Critical Node Cut *has a polynomial kernel with respect to* $k + x$*.*

3 Parameter w

In this section we will show that CNC is unlikely to be fixed-parameter tractable when parameterized by w. This implies that we cannot substantially improve on the $O(n^{w+1})$ algorithm of [1]. Since we will not directly use the notion of treewidth and tree decompositions, we refer to [4] for their definition.

▶ **Theorem 5.** Critical Node Cut *is* W[1]*-hard with respect to the treewidth* w *of the input graph.*

Our proof of the theorem above is via the well-known multicolored clique technique [17] which utilizes generic gadget structure to construct a reduction from the W[1]-complete Multicolored Clique problem: Given an undirected simple graph G with n vertices and m edges, a coloring function $c : V(G) \to \{1, \ldots, \ell\}$ of the vertices of G, and a parameter ℓ, determine whether G has a clique which includes exactly one vertex from each color. Throughout this section we use (G, c, ℓ) to denote an arbitrary input to Multicolored Clique. As usual in parameterized reductions, we can assume that n and ℓ are sufficiently larger than any fixed constant, and that n is sufficiently larger than ℓ.

In the multicolored clique technique, we construct *selection* gadgets which encode the selection of vertices and edges of G (one per each color class and pair of color classes, respectively), and *validation* gadgets which ensure that the vertices and edges selected indeed form a clique in G. In our reduction below, we will force any feasible solution to delete a large number of vertices from the constructed CNC instance in order to reach the required bound on the number of remaining connected pairs. We will ensure that such a solution always leaves $4\binom{\ell}{2}$ very large components which encode the selection of $\binom{\ell}{2}$ edges in G. The bound on the number of connected pairs will require all these huge components to have equal size, which in turn can only happen if the edges selected in G are edges between the same set of ℓ vertices (implying that these ℓ vertices form a clique in G). In what follows, we use (H, k, x) to denote the instance of CNC that we construct, where H is the input graph, k is the size of the required cut, and x is the bound on the number of connected pairs. Note that for our proof to go through, we will also need to show that the treewidth of H is bounded by some function in ℓ.

Connector gadgets. To each vertex $u \in V(G)$, we assign two unique integer identifiers: $low(u) \in \{1, \ldots, n\}$ and $high(u) \in \{n+1, \ldots, 2n\}$, where $high(u) = 2n + 1 - low(u)$. Our selection gadgets are composed from gadgets which we call *connector gadgets*. A connector gadget *corresponds* to a vertex of G, and can be of *low order* or *high order*. A low order connector gadget corresponding to a vertex $u \in V(G)$ consists of a clique of size ℓ^4 and an independent set of size $n^{16} + low(u)$ which have all edges between them; *i.e.*, it is a complete split graph on these two sets of vertices. Similarly, a high order connector gadget corresponding to $u \in V(G)$ is a complete split graph on a clique of size ℓ^4 and an independent set of size $n^{16} + high(u)$.

We refer to the clique in a connector gadget as the *core* of the gadget, and to the remaining vertices as the *guard* of the gadget. Only vertices in the core will be adjacent to vertices outside the gadget. Notice that the huge independent set in the core contributes to a large number of connected pairs in H, and one can delete all these connected pairs only by adding all core vertices to the solution cut. Below we use this property to help us control solutions for our CNC instance.

Selection gadgets. The graph H consists of a selection gadget for each vertex and edge in G (see Figure 1): For a vertex $u \in V(G)$, we will construct a *u-selection gadget* as follows: First we add a clique U of size ℓ^2 to H, and then we connect all the vertices of U to an additional independent set of n^9 vertices, which we call the *dummy vertices* of the u-selection gadget. We next connect U to $(\ell - 1)$ gadget pairs, one pair for each color $i \in \{1, \ldots, \ell\} \setminus \{c(u)\}$. Each pair consists of a low order and a high order connector gadget corresponding to u. We let $A_o^i[u]$ and $B_o^i[u]$ respectively denote the core and guard of the connector gadget

associated with color $i \in \{1,\ldots,\ell\} \setminus \{c(u)\}$ and of order $o \in \{low, high\}$. We connect U to each connector gadget by adding all edges between all vertices of U and $A_o^i[u]$, for each $i \in \{1,\ldots,\ell\} \setminus \{c(u)\}$ and $o \in \{low, high\}$.

For an edge $\{u_1, u_2\} \in E(G)$, we will construct a $\{u_1, u_2\}$-*selection gadget* as follows: First we add a vertex which we denote by $\{u_1, u_2\}$ to H. We then connect $\{u_1, u_2\} \in V(H)$ to a low order and a high order connector gadget associated with u_1, and to a low order and a high order connector gadget associated with u_2, by adding all edges between vertex $\{u_1, u_2\} \in V(H)$ and the core vertices of these gadgets. We let $A_o^u[u_1, u_2]$ and $B_o^u[u_1, u_2]$ respectively denote the core and guard of the connector gadget corresponding to $u \in \{u_1, u_2\}$ of order $o \in \{low, high\}$ in the $\{u_1, u_2\}$-selection gadget. Finally, we connect $\{u_1, u_2\} \in V(H)$ to an additional set of n^4 dummy neighbors of degree one in H.

Validation gadgets. We next add the validation gadgets to H, one for each ordered pair of distinct colors (i, j), $i \neq j$. For such a pair (i, j), the (i, j)-validation gadget simply consists of two cliques $V_{low}[i, j]$ and $V_{high}[i, j]$, each of size ℓ^7. The validation is done through the connections of these two cliques to the remainder of the graph. Consider a u-selection gadget for a vertex $u \in V(G)$ of color i. We add all possible edges between $V_{low}[i, j]$ and $A_{low}^j[u]$, and all edges between $V_{high}[i, j]$ and $A_{high}^j[u]$. This is done for every vertex of color i. Consider next a $\{u_1, u_2\}$-selection gadget where $c(u_1) = i$ and $c(u_2) = j$. We add all possible edges between $V_{low}[i, j]$ and $A_{high}^{u_1}[u_1, u_2]$, and all possible edges between $V_{high}[i, j]$ and $A_{low}^{u_1}[u_1, u_2]$. In this way, $V_{low}[i, j]$ is connected to low order connector gadgets of vertex selection gadgets and to high order connector gadgets of edge selection gadgets, and $V_{high}[i, j]$ is connected in the opposite way.

CNC instance. The graph H of our CNC instance is thus composed of $4\binom{\ell}{2}$ validation cliques which have ℓ^7 vertices each, n vertex selection gadgets each of size $(\ell-1)(2n^{16}+2n+1+2\ell^4)+n^9+\ell^2$, and m edge selection gadgets which have $2(2n^{16}+2n+1+2\ell^4)+n^4+1$ vertices each. We finish the description of our reduction by setting k, the size of the required critical node cut, to

$$k := \left(2(\ell-1)n + 4m - 8\binom{\ell}{2}\right) \cdot \ell^4 + \ell^3 + \binom{\ell}{2},$$

and setting x, the bound on the number of connected pairs, to

$$\begin{aligned} x \;:=\; & (n-\ell)\,(n^9+\ell^2)(n^9+\ell^2-1) + \left(m - \binom{\ell}{2}\right)(n^4+1)n^4 + \\ & 4\binom{\ell}{2}(2n^{16}+2n+1+\ell^7+2\ell^4)(2n^{16}+2n+\ell^7+2\ell^4). \end{aligned}$$

▶ **Lemma 6.** *The graph H has treewidth at most $4\binom{\ell}{2}\ell^7 + \ell^4 + \ell^2$.*

Proof. We use two well known facts about treewidth: The treewidth of a graph is the maximum treewidth of all its components, and adding α vertices to a graph of treewidth at most β results in a graph of treewidth at most $\alpha + \beta$. Using these two facts we get that a connector gadget has treewidth at most ℓ^4, since we add ℓ^4 vertices to a graph of treewidth 0 (the independent set of vertices). From this we conclude that each selection gadget has treewidth at most $\ell^4 + \ell^2$, since we either add a clique of size ℓ^2 or a single vertex to a graph whose connected components have treewidth bounded by ℓ^4. Therefore, since H itself is constructed by adding $4\binom{\ell}{2} \cdot \ell^7$ validation vertices to a graph whose connected components have treewidth at most $\ell^4 + \ell^2$, the lemma follows. ◀

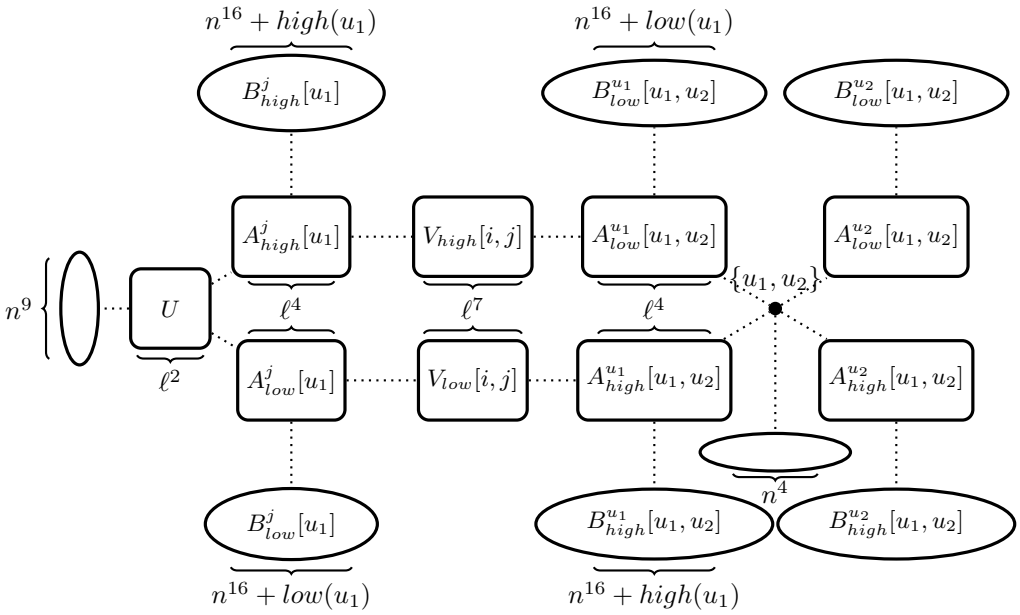


Figure 1 The connection of selection gadgets via a validation gadget. In the example, we consider a vertex $u_1 \in V(H)$ with $c(u_1) = i$ which is adjacent to a vertex $u_2 \in V(H)$ with $c(u_2) = j$. The diagram depicts the pair of low and high connector gadgets associated with color j in the u-selection gadget that are connected to the $\{u_1, u_2\}$-selection gadget. The remaining $(\ell - 2)$ pairs of connector gadgets in the u-selection gadget are not depicted. The rectangle boxes represent cliques and each ellipsis represents an independent set. The dotted lines depict a complete set of edges between two sets of vertices.

From a multicolored clique to a critical node cut. Suppose (G, c, ℓ) has a solution, *i.e.*, a multicolored clique S of size ℓ. Then one can verify that the cut $C \subseteq V(H)$ defined by

$$C := \{U : u \in S\} \cup \{\{u_1, u_2\} : u_1 \neq u_2 \in S\} \cup \big\{v : v \in A^c_o[u], u \notin S\big\} \cup \big\{v : v \in A^u_o[u_1, u_2], u_1 \neq u_2 \notin S\big\}$$

is of size k, and $H - C$ contains exactly three types of non-trivial connected components:

- $n - \ell$ components which include a clique U of size ℓ^2 along with n^9 dummy vertices.
- $m - \binom{\ell}{2}$ components which include a single vertex of $E(G)$ along with n^4 dummy vertices.
- $4\binom{\ell}{2}$ components which have $2n^{16} + 2n + 1 + \ell^7 + 2\ell^4$ vertices each.

Thus, $H - C$ has exactly x connected pairs, and C is indeed a solution to (H, k, x).

From a critical node cut to a multicolored clique. To complete the proof of Theorem 5, we show that if (H, k, x) has a solution, *i.e.*, a cut C of size k where $H - C$ has at most x connected pairs, then G has a multicolored clique of size ℓ. We do this, using a few lemmas that restrict the structure of solutions to our CNC instance. The first one of these, Lemma 7 below, shows that we can restrict our attention to cuts which include only core vertices of connector gadgets and vertices of $V(G) \cup E(G)$.

▶ **Lemma 7.** *If there is a solution to (H, k, x), then there is a solution C to this instance which includes no guard vertices, no dummy vertices, and no validation vertices of H.*

Proof. Let C be a solution to (H, k, x). If C includes any dummy vertex v of H, then since v is a vertex whose neighborhood is a clique, we can either replace v with one of its neighbors (which is a non-dummy vertex) or, if C contains all neighbors of v, we remove v from C. Both modifications of C do not increase the number of connected pairs in $H - C$. Similarly, if C includes guard vertices, these can be safely replaced with core vertices.

Next, we show that C cannot contain any validation clique completely. To this end, note that a core of a connector gadget which is not completely included in C contributes more than n^{32} connected pairs in $H - C$. This can be seen by counting the number of connected pairs between a single core vertex and all of its guard neighbors. Thus, since $(16\binom{\ell}{2}+1)n^{32} > x$ assuming a sufficiently large n, the cut C must include all but at most $16\binom{\ell}{2}$ cores of connector gadgets in H. But as each validation clique is of size $\ell^7 > 8\binom{\ell}{2}\ell^4 + \ell^3 + \binom{\ell}{2}$ (for sufficiently large ℓ), we have $k - \ell^7 < (2(\ell-1)n + 4m - 16\binom{\ell}{2})\ell^4$, which means that if C includes a validation clique it does not include enough cores. Thus, C cannot completely contain any validation clique.

Finally, consider the case that C contains a proper subset of some validation clique $V_o[i,j]$ in H. Observe first that if the validation clique is not completely isolated in $H - C$, then a vertex $v \in C \cap V_o[i,j]$ can be safely replaced by a core vertex that is adjacent to $V_o[i,j]$ as v is not a cut vertex in $H - (C \setminus \{v\})$. Thus, the only remaining case is that all vertices that have a neighbor in $V_o[i,j]$ are in C. Then, deleting the vertices in $V_o[i,j]$ removes at most $\ell^7(\ell^7 - 1)$ connected pairs. By the choice of k, and the number of core vertices, C cannot contain all core vertices. Consider a core vertex $u \notin C$. Since C does not contain any guard vertices, adding u to C removes at least $n^{16} > \ell^7(\ell^7 - 1)$ connected pairs. Thus, we can remove all vertices of $V_o[i,j] \cap C$ from C and replace them by u without increasing the number of connected pairs in $H - C$. Thus, there is a solution that contains no vertices of validation cliques. ◀

Assume that (H, k, x) has a solution, and fix a solution C as in Lemma 7. By the definition of k, we know that the cut C cannot include all connector gadgets. A connector gadget in $H - C$ induces a large number of connected pairs, at least n^{32}, due to the guard vertices of the gadget. Let us therefore call a connected component in $H - C$ *huge* if it contains at least n^{32} connected pairs. The next lemma shows that there can only be a certain number of these huge components in $H - C$, and reveals some further restriction on any solution cut C. We call a maximal non-empty (but not necessarily proper) subset of a core in $H - C$ a *partial core*.

▶ **Lemma 8.** *If C is a solution to (H, k, x) as in Lemma 7, then C includes $(2(\ell-1)n + 4m - 8\binom{\ell}{2})$ cores. Furthermore, there are precisely $4\binom{\ell}{2}$ huge components in $H - C$, each consisting of a validation clique, two partial cores, and the two guard sets of the partial cores.*

Proof. Let $A_1, \ldots, A_t$ denote all partial cores in $H - C$. Note that since each core is of size $\ell^4 > \ell^3 + \binom{\ell}{2}$ (for sufficiently large ℓ), the cut C can include at most $(2(\ell-1)n + 4m - 8\binom{\ell}{2})$ complete cores by definition of k, and so $t \geq 8\binom{\ell}{2}$. By Lemma 7, the graph $H - C$ contains all $4\binom{\ell}{2}$ validation cliques. Let $Q_1, \ldots, Q_s$ denote the components in $H - C$ that contain at least one validation clique, and let $q_i := |Q_i| - 1$ for each i, $1 \leq i \leq s$. Observe that for any huge component Q in $H - C$, we have $Q \in \{Q_1, \ldots, Q_s\}$.

Now, since the total number of validation cliques is $4\binom{\ell}{2}$, we have $s \leq 4\binom{\ell}{2}$, and the total number of connected pairs in all the Q_i's is lower bounded by $\sum_{i=1}^{s} q_i^2$. Note that each partial core A_j belongs to some Q_i and contributes at least $n^{16} + 1$ vertices to its size (accounting for a single vertex of A_j and all its guard neighbors), and therefore at least n^{32} connected pairs. It can now be seen that since $\sum_{i=1}^{s} q_i^2$ is concave and symmetric, it is

minimized when the number of addends is as large as possible and all of the addends are of equal size. This happens when $s = 4\binom{\ell}{2}$ and each Q_i includes exactly two A_j's, giving us $\sum_{i=1}^{s} q_i^2 = \sum_{i=1}^{s}((2n^{16})^2) + o(n^{32}) = 16\binom{\ell}{2}n^{32} + o(n^{32})$. If $s < 4\binom{\ell}{2}$ or there is one Q_i that contains more then two A_j's, then the sum will be at least $(16\binom{\ell}{2}+1)n^{32} > x$. It follows that there are exactly $4\binom{\ell}{2}$ huge components, that each have two A_j's. These huge components contribute altogether at least $16\binom{\ell}{2}n^{32}$ connected pairs.

We have thus established that there are $4\binom{\ell}{2}$ huge components in $H - C$, and each includes a validation clique, two partial cores, and the guard sets adjacent to these partial cores which are not in C according to Lemma 7. To see that the huge components contain nothing else, recall first that the overall number of connected pairs in these huge components is at least $16\binom{\ell}{2}n^{32}$. Thus, the number of further additional connected pairs in $H - C$ is at most $x - 16\binom{\ell}{2}n^{32} = n \cdot n^{18} + o(n^{18}) < 2n^{19}$. Now, if A contains other vertices, then by construction it must contain either a vertex from a clique U corresponding to a vertex u of G, or a vertex $\{u, u'\}$ corresponding to an edge of G. In either of these cases, this additional vertex is adjacent to at least n^4 dummy vertices, implying that Q has an additional number of $n^4 \cdot n^{16} = n^{20} > 2n^{19}$ connected pairs, a contradiction. ◀

Slightly smaller than huge components are *large components* in $H - C$ which have at least n^{18} connected pairs and fewer than n^{32} connected pairs. Further smaller are *big components* which have at least n^8 connected pairs, and less than n^{18} connected pairs.

▶ **Lemma 9.** *If C is a solution to (H, k, x) as in Lemma 7, then C includes exactly ℓ cliques $U_1, \ldots, U_\ell$ corresponding to vertices $u_1, \ldots, u_\ell \in V(G)$, and there are precisely $n - \ell$ large components in $H - C$.*

Proof. Note that $x = 16\binom{\ell}{2}n^{32} + (n - \ell)n^{18} + o(n^{18})$. By Lemma 8, we know that $H - C$ contains $4\binom{\ell}{2}$ huge components, and so these already account for $16\binom{\ell}{2}n^{32}$ connected pairs in $H - C$. For every $u \in V(G)$, if the clique corresponding to U is not completely contained in C, then there is a large component corresponding to u in $H - C$, since by Lemma 7, all n^9 dummy neighbors of U are existent in $H - C$. Furthermore, any large component in $H - C$ is of this form. Thus, if C contains $\ell' < \ell$ cliques corresponding to vertices of G, then the number of connected pairs in $H - C$ is at least $16\binom{\ell}{2}n^{32} + (n - \ell')n^{18} > 16\binom{\ell}{2}n^{32} + (n - \ell)n^{18} + o(n^{18}) = x$, a contradiction. Moreover, by our choice of k, the cut C cannot include $(2(\ell - 1)n + 4m - 8\binom{\ell}{2})$ cores (as is necessary by Lemma 8) and more than ℓ such cliques U, since $(2(\ell - 1)n + 4m - 8\binom{\ell}{2})\ell^4 + (\ell + 1)\ell^2 > k$. ◀

▶ **Lemma 10.** *If C is a solution to (H, k, x) as in Lemma 7, then C includes exactly $\binom{\ell}{2}$ vertices which correspond to edges in G, and there are precisely $m - \binom{\ell}{2}$ big components in $H - C$.*

Proof. Let us call each element in the set $\{U \subset V(H) : u \in V(G)\} \cup \{\{u, u'\} \in V(H) : \{u, u'\} \in E(G)\}$ a *G-element*. Thus, each G-element belongs to its unique selection gadget in H, and corresponds to either a vertex or an edge of G. Moreover, each core is adjacent to exactly one G-element. By Lemma 9 we know that C contains ℓ G-elements corresponding to vertices of G. We next argue that it also contains $\binom{\ell}{2}$ G-elements corresponding to edges of G.

Consider a huge component Q in $H - C$. By Lemma 8, Q contains two partial cores A and A' and Q does *not* contain the unique G-element that is adjacent to the two partial cores. Thus, the G-element neighbors of exactly $8\binom{\ell}{2}$ partial cores are contained in C. The set of cliques $U_1, \ldots, U_\ell \subseteq C$ promised by Lemma 9 accounts for at most $2(\ell - 1) \cdot \ell = 4\binom{\ell}{2}$ such cores, as each U_i has exactly $2(\ell - 1)$ neighboring cores in H. Notice that by the choice of k,

after accounting for the vertices in C required by Lemma 8 and Lemma 9, the remaining number of vertices is $\binom{\ell}{2}$. Any G-element representing a vertex has $\ell^2 > \binom{\ell}{2}$ vertices, and thus all remaining deleted G-elements correspond to edges of G. Now observe that each of them can account for at most four partial cores as they have exactly four neighboring cores in H. Consequently, the number of deleted G-elements that correspond to edges in G is at least $\binom{\ell}{2}$. By the choice of k, it is thus exactly $\binom{\ell}{2}$. ◀

▶ **Lemma 11.** *The set of vertices $u_1, \ldots, u_\ell$ specified in Lemma 9 induces a multicolored clique in G.*

Proof. Lemma 8, Lemma 9, and Lemma 10 together state that C includes at least $(2(\ell-1)n + 4m - 8\binom{\ell}{2}) \cdot \ell^4$ core vertices, at least ℓ^3 vertices in cliques corresponding to vertices of G, and at least $\binom{\ell}{2}$ vertices corresponding to edges of G. By our selection of k, all these lower bounds are in fact equalities. Thus, all but ℓ cliques U, $u \in V(G)$, are present in $H - C$, and all but $\binom{\ell}{2}$ edges of G are present in $H - C$. All these vertices contribute at least $(n-\ell)\,(n^9+\ell^2)(n^9+\ell^2-1) + \left(m - \binom{\ell}{2}\right)(n^4+1)n^4$ connected pairs in $H - C$, due to their dummy neighbors. Thus, by definition of x, the total number of connected pairs from huge components in $H - C$ is $4\binom{\ell}{2}(2n^{16}+2n+1+\ell^7+2\ell^4)(2n^{16}+2n+\ell^7+2\ell^4)$.

Now, note that according to Lemma 8, $H - C$ includes exactly $8\binom{\ell}{2}$ partial cores with no neighboring G-elements. The set of cliques $U_1, \ldots, U_\ell \subseteq C$, promised by Lemma 9, accounts for at most $2(\ell-1)\cdot\ell = 4\binom{\ell}{2}$ partial cores. Moreover, each clique (corresponding to a vertex) is of a different color, otherwise the specific structure promised by Lemma 8 is violated. Similarly, the $\binom{\ell}{2}$ deleted G-elements that correspond to edges in G, promised by Lemma 10, account for at most $4\binom{\ell}{2}$ partial cores, and each edge corresponds to a different pair of colors. Consequently, the only way to remove the required number of neighboring G-elements is if these upper bounds are met with equality. Thus, we have ℓ vertices and $\binom{\ell}{2}$ edges of different colors, as required in a multicolored clique.

Finally, observe that due to the fact that we have accounted for all the vertices in C, it is clear that each huge component consists of two complete (*i.e.*, non-partial) cores. Thus, the size of each of these huge components is $2n^{16}+\ell^7+2\ell^4+high(u_1)+low(u_1')$ for $u_1, u_1' \in V(G)$. Therefore, the only way for the total number of connected pairs in all huge components to not exceed $4\binom{\ell}{2}(2n^{16}+2n+1+\ell^7+2\ell^4)(2n^{16}+2n+\ell^7+2\ell^4)$ is if all huge components have equal size, *i.e.*, exactly $(2n^{16}+2n+1+\ell^7+2\ell^4)$ vertices each. But this can happen only if we have $u_1 = u_1'$ in the pair of connector guards $B_o^i[u_1]$ and $B_{\bar{o}}^{u_1'}[u_1', u_2]$, in each huge component of $H-C$, as this is the only way for the guard vertices to sum up to $2n^{16}+2n+1$. Consequently, the set of $\binom{\ell}{2}$ edges selected in G are edges between $u_1, \ldots, u_\ell$ implying that they indeed form a clique. ◀

4 Parameters $w + x$ and y

If we combine the treewidth parameter w with the parameter for the number of connected pairs x, then we obtain fixed-parameter tractability. This can be derived via an optimization variant of Courcelle's theorem due to Arnborg et al. [2]. Using tree decompositions, we obtain a more efficient algorithm.

▶ **Theorem 12.** *The* Critical Node Cut *problem is* FPT *with respect to $w + x$.*

Finally, we consider the CNC problem parameterized by y. We will show that the problem is FPT under this parameterization but has no polynomial kernel even for the aggregate parameterization of $k + y + w$.

▶ **Theorem 13.** *The* Critical Node Cut *problem is FPT with respect to* y.

▶ **Theorem 14.** *The* Critical Node Cut *problem parameterized by* $k + y + w$ *has no polynomial kernel unless the polynomial hierarchy collapses.*

5 Discussion

We considered a natural graph cut problem called Critical Node Cut (CNC) under the framework of parameterized complexity. The only parameterization left open in our analysis is the parameter $w + k$, and so the first natural question left open in the paper is whether CNC is fixed-parameter tractable under this parameterization (we know it is unlikely that it admits a polynomial kernel). It would also be interesting to see how parameters *maximum degree* and *pathwidth* affect the parameterized complexity of CNC. Finally, one can consider the edge variant of the problem (where one is required to delete edges instead of vertices) and the directed variant of the problem. Several of our proofs do not transfer directly to these variants. For example, it is open whether the edge deletion variant of CNC is W[1]-hard with respect to the number of edge deletions.

Acknowledgments. The research leading to these results has received funding from the People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme (FP7/2007-2013) under REA grant agreement number 631163.11, and by the ISRAEL SCIENCE FOUNDATION (grant No. 551145/).

References

1 Bernardetta Addis, Marco Di Summa, and Andrea Grosso. Removing critical nodes from a graph: complexity results and polynomial algorithms for the case of bounded treewidth. *Optimization online (www.optimization-online.org)*, 2011.

2 Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991.

3 Ashwin Arulselvan, Clayton W Commander, Lily Elefteriadou, and Panos M Pardalos. Detecting critical nodes in sparse graphs. *Computers & Operations Research*, 36(7):2193–2200, 2009.

4 Hans L Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11(1-2):1, 1994.

5 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.

6 Vladimir Boginski and Clayton W Commander. Identifying critical nodes in protein–protein interaction networks. *Clustering challenges in biological networks*, pages 153–167, 2009.

7 Nicolas Bousquet, Jean Daligault, and Stéphan Thomassé. Multicut is FPT. In *Proceedings of the 43rd Annual Symposium on Theory of Computing (STOC)*, pages 459–468. ACM, 2011.

8 Karl Bringmann, Danny Hermelin, Matthias Mnich, and Erik Jan van Leeuwen. Parameterized complexity dichotomy for steiner multicut. In *Proceedings of the 32nd Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 157–170, 2015.

9 Yixin Cao, Jianer Chen, and Jia-Hao Fan. An $O(1.84^k)$ parameterized algorithm for the multiterminal cut problem. In *Proceedings of the 19th Annual Symposium on Fundamentals of Computation Theory (FCT)*, pages 84–94, 2013.

10 Jianer Chen, Yang Liu, and Songjian Lu. An improved parameterized algorithm for the minimum node multiway cut problem. *Algorithmica*, 55(1):1–13, 2009.

11 Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. On multiway cut parameterized above lower bounds. *ACM Transactions on Computation Theory*, 5(1):3, 2013.

12 Marco Di Summa, Andrea Grosso, and Marco Locatelli. Complexity of the critical node problem over trees. *Computers & Operations Research*, 38(12):1766–1774, 2011.

13 Rodney G Downey and Michael R Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.

14 Pål Grønås Drange, Markus Sortland Dregi, and Pim van't Hof. On the computational complexity of vertex integrity and component order connectivity. In *Proceedings of the 25th Annual International Symposium on Algorithms and Computation (ISAAC)*, pages 285–297, 2014.

15 Andrew Drucker. New limits to classical and quantum instance compression. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2012.

16 Jessica Enright and Kitty Meeks. Deleting edges to restrict the size of an epidemic. *CoRR*, abs/1504.05773, 2015.

17 Michael R Fellows, Danny Hermelin, Frances A Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theoretical Computer Science*, 410(1):53–61, 2009.

18 Michael R Fellows and Sam Stueckle. The immersion order, forbidden subgraphs and the complexity of network integrity. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 6(1):23–32, 1989.

19 Michael R Garey and David S Johnson. Computers and intractability; a guide to the theory of NP-completeness. *San Francisco, LA: Freeman*, 1979.

20 Sylvain Guillemot. FPT algorithms for path-transversal and cycle-transversal problems. *Discrete Optimization*, 8(1):61–71, 2011.

21 Karen E Joyce, Paul J Laurienti, Jonathan H Burdette, and Satoru Hayasaka. A new measure of centrality for brain networks. *PLoS One*, 5(8):e12200, 2010.

22 Stefan Kratsch, Marcin Pilipczuk, Michal Pilipczuk, and Magnus Wahlström. Fixed-parameter tractability of multicut in directed acyclic graphs. *SIAM Journal on Discrete Mathematics*, 29(1):122–144, 2015.

23 Vito Latora and Massimo Marchiori. How the science of complex networks can help developing strategies against terrorism. *Chaos, solitons & fractals*, 20(1):69–75, 2004.

24 Dániel Marx. Parameterized graph separation problems. *Theoretical Computer Science*, 351(3):394–406, 2006.

25 Dániel Marx, Barry O'Sullivan, and Igor Razgon. Finding small separators in linear time via treewidth reduction. *ACM Transaction on Algorithms*, 9(4):1–30, 2013.

26 Dániel Marx and Igor Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset. In *Proceedings of the 43rd Annual Symposium on Theory of Computing (STOC)*, pages 469–478, 2011.

27 Cullen M Taniguchi, Brice Emanuelli, and C Ronald Kahn. Critical nodes in signalling pathways: insights into insulin action. *Nature Reviews Molecular Cell Biology*, 7(2):85–96, 2006.

28 Mario Ventresca. Global search algorithms using a combinatorial unranking-based problem representation for the critical node detection problem. *Computers & Operations Research*, 39(11):2763–2775, 2012.

29 Mario Ventresca and Dionne Aleman. A derandomized approximation algorithm for the critical node detection problem. *Computers & Operations Research*, 43:261–270, 2014.

30 Mingyu Xiao. Simple and improved parameterized algorithms for multiterminal cuts. *Theory of Computing Systems*, 46(4):723–736, 2010.

Parameterized Complexity of Sparse Linear Complementarity Problems

Hanna Sumita[1], Naonori Kakimura[2], and Kazuhisa Makino[3]

1 **JST, ERATO, Kawarabayashi Large Graph Project,**
National Institute of Informatics
Tokyo 101-8430, Japan
sumita@nii.ac.jp

2 **Graduate School of Arts and Sciences, University of Tokyo**
Tokyo 153-8902, Japan
kakimura@global.c.u-tokyo.ac.jp

3 **Research Institute for Mathematical Sciences, Kyoto University**
Kyoto 606-8502, Japan
makino@kurims.kyoto-u.ac.jp

Abstract

In this paper, we study the parameterized complexity of the linear complementarity problem (LCP), which is one of the most fundamental mathematical optimization problems. The parameters we focus on are the sparsities of the input and the output of the LCP: the maximum numbers of nonzero entries per row/column in the coefficient matrix and the number of nonzero entries in a solution. Our main result is to present a fixed-parameter algorithm for the LCP with all the parameters. We also show that if we drop any of the three parameters, then the LCP is fixed-parameter intractable. In addition, we discuss the nonexistence of a polynomial kernel for the LCP.

1998 ACM Subject Classification G.1.6 Optimization

Keywords and phrases linear complementarity problem, sparsity, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.355

1 Introduction

Given a square matrix $M \in \mathbb{R}^{n\times n}$ and a vector $q \in \mathbb{R}^n$, the *linear complementarity problem* (LCP) is to find a vector $z \in \mathbb{R}^n$ such that

$$Mz + q \geq 0, \quad z \geq 0, \quad z^\top (Mz + q) = 0. \tag{1}$$

We denote a problem instance of the LCP with M and q by LCP (M, q). We say that n is the *order* of LCP (M, q). The LCP, introduced by Cottle [9], Cottle and Dantzig [10], and Lemke [24], is one of the most widely studied mathematical programming problems, which, for example, contains linear and convex quadratic programming problems. The decision version of the LCP (i.e., deciding whether (1) has a solution z) is NP-complete [6]. For details of the LCP and related topics, see the books of Cottle, Pang, and Stone [11] and Murty [25].

In this paper, we analyze the parameterized complexity of the LCP. A problem with parameter k is said to be *fixed-parameter tractable* if there exists an algorithm which solves the problem in $f(k) \cdot L^{\mathrm{O}(1)}$ time, where f is some computable function and L is the size of a

10th International Symposium on Parameterized and Exact Computation (IPEC 2015).
Editors: Thore Husfeldt and Iyad Kanj; pp. 355–364

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

given instance. An algorithm with such running time is called a *fixed-parameter algorithm*. See e.g., [15, 17] for the detail of the parameterized complexity theory.

The parameterized complexity has been recently attracting attention in the field of mathematical programming, as optimization problems in the real world often have a certain parameterized structure. In particular, sparsities of the input and the output have been investigated, motivated from other fields such as computational biology [12] and coding theory [1]. In this paper, we focus on the sparsities of the input and the output of the LCP.

We consider as parameters the numbers of nonzero entries per row and column in an input matrix. We say that the LCP is *r-row-sparse* (resp. *c-column-sparse*) if the coefficient matrix has at most r nonzero entries per row (resp. at most c nonzero entries per column). The LCP is *(r, c)-sparse* if it is r-row-sparse and c-column-sparse. It is not difficult to see that any LCP instance can be reduced to a 3-row-sparse LCP instance (or a 3-column-sparse LCP instance), and thus it is NP-hard if $r = 3$ or $c = 3$. Moreover, even the 2-row-sparse LCP is NP-hard [27], while the 1-row-sparse LCP is linear-time solvable. Let us remark that this kind of sparsity has been studied for the special classes of the LCP, i.e., the system of linear inequalities [8, 20] and the bimatrix game [4, 5, 7, 13, 19]. The *ℓ-sparse* bimatrix game has payoff matrices each of whose rows and columns has at most ℓ nonzero entries, which can be described as the (ℓ, ℓ)-sparse LCP when a mixed Nash equilibrium has a positive expected payoff. It is also known that the mean payoff game falls into the $(3, c)$-sparse LCP [2], where c is the maximum indegree of a given graph plus two.

Another parameter discussed in this paper is the size of the *support* of a solution, that is, the number of nonzero entries of an output solution. For an integer s, the *s-support* LCP is to find a solution of (1) with support size at most s. Finding a solution with small support size is often useful in applications. For example, in the contact problem of rigid bodies (see e.g., [11]), which is formulated as the LCP, a solution is usually required to have a few nonzero entries when the bodies are in contact at few points on the surfaces. For the bimatrix game, which is also a special class of the LCP, a mixed Nash equilibrium with small support size has been studied recently [16, 19], as such a solution is close to a pure Nash equilibrium.

Our main result is to present a fixed-parameter algorithm for the s-support (r, c)-sparse LCP with parameters s, r, and c. The running time is $((r + c)rcs)^{\mathrm{O}(cs)} \cdot n^{\mathrm{O}(1)}$ time, where n is the order of a given instance. To obtain the algorithm, we incorporate the idea of Hermelin, Huang, Kratsch and Wahlström [19] for the bimatrix game. We first construct a graph associated with a given LCP instance, and derive a necessary condition on the graph that the support of a solution must satisfy. Our algorithm enumerates all index sets satisfying the necessary condition by traversing the graph. Since the number of candidate index sets is bounded by $((r + c)rcs)^{\mathrm{O}(cs)}$, the algorithm runs in fixed-parameter time.

On the other hand, we show that, if we drop any of the parameters r, c and s, then the LCP becomes fixed-parameter intractable. We first prove the $W[1]$-hardness of the s-support r-row-sparse LCP (resp., the s-support c-column-sparse LCP) with parameters s, r (resp., s, c). We remark that the s-support LCP with only parameter s is $W[2]$-hard by reducing the problem of finding a Nash equilibrium with support size at most s of the bimatrix game [16].

For the case when we drop the support size s of a solution, the $(2, 2)$-sparse LCP is shown to be NP-hard by extending an NP-hardness proof of the 2-row-sparse LCP [27]. We remark that the 1-row-sparse LCP (resp., the 1-column-sparse LCP) is solvable in linear time [27]. Thus it reveals the time complexity of the (r, c)-sparse LCP with respect to r and c.

Table 1 summarizes our results.

We also show the nonexistence of a polynomial kernel for the s-support (r, c)-sparse LCP unless coNP $\subseteq$ NP/poly. A *kernelization* algorithm is a polynomial-time transformation of a

Table 1 The parametrized complexity of sparse LCPs.

Input \ Output	general	support size s
general	NP-hard [6]	$W[2]$-hard [16]
row-sparsity r	NP-hard ($r = 2$) [27]	$\boldsymbol{W[1]}$**-hard** (Theorem 7)
column-sparsity c	**NP-hard ($c = 2$)** (Theorem 10)	$\boldsymbol{W[1]}$**-hard** (Theorem 8)
r and c	**NP-hard ($r = c = 2$)** (Theorem 10)	$\boldsymbol{((r+c)rcs)^{O(cs)} \cdot n^{O(1)}}$ (Theorem 6)

problem instance to an equivalent instance whose size and parameter depend only on the parameter of the original instance. The output of a kernelization algorithm is called a *kernel*, and a *polynomial kernel* if the size and the parameter are bounded by a polynomial in the original parameter. Recently, the existence of (polynomial) kernels has been investigated for mathematical programming problems such as a system of sparse linear equations [12] and integer linear programming problems [21, 22, 23]. While the s-support (r, c)-sparse LCP is fixed-parameter tractable, we show that a polynomial kernel does not exist by applying a general framework by Bodlaender, Downey, Fellows and Hermelin [3].

We remark that our hardness results are not implied directly by those on the sparse bimatrix game [4, 5, 19], since a natural reduction to the LCP (see e.g., [11]) destroys the sparsity.

We summarize the notation used in this paper. For a positive integer n, let $[n] = \{1, \ldots, n\}$. For a set $B \subseteq [n]$, we define $\overline{B} = [n] \setminus B$. Let M be an $m \times n$ real matrix, where M has a row index set $[m]$ and a column index set $[n]$. For $S \subseteq [m]$ and $T \subseteq [n]$, we denote by M_{ST} the submatrix of M such that S and T are row and column index sets, respectively. We also define $M_{\cdot T}$ by $M_{\cdot T} = M_{[m]T}$. If $S = \{i\}$, we simply write $M_{i\cdot}$, M_{iT} and M_{ij} instead of $M_{\{i\}[n]}$, $M_{\{i\}T}$ and $M_{\{i\}\{j\}}$, respectively. Let z be a vector in $\mathbb{R}^n$ with index set $[n]$. For index set $B \subseteq [n]$, let z_B denote the subvector of z with entries corresponding to B. For $i \in [n]$, we also denote by z_i the i-th entry of z.

The rest of the paper is organized as follows. Section 2 presents a fixed-parameter algorithm for the parameterized LCP. Section 3 shows the fixed-parameter intractability of the LCP. Section 4 discusses the nonexistence of a polynomial kernel. Some proofs are omitted due to the space limitation.

2 Fixed-parameter algorithm

In the section, we present a fixed-parameter algorithm for the s-support (r, c)-sparse LCP with parameter $s + r + c$. Let LCP (M, q) be an (r, c)-sparse LCP instance of order n.

▶ **Definition 1.** For a vector x, the *support* of x is the index set $\{i \mid x_i > 0\}$, denoted by S_x.

It is observed that a solution z of LCP (M, q) is a feasible solution to the system of linear inequalities

$$Mz + q \geq 0,\ z \geq 0,\ z_{\overline{S}} = 0,\ \text{and}\ (Mz + q)_S = 0. \tag{2}$$

Conversely, if we have a vector z satisfying (2), then z is a solution of LCP (M, q). Thus if we know the support S of some solution, we can find a solution z with $S_z \subseteq S$ in polynomial

time by solving the system of linear inequalities (2). Hence, enumerating all the index sets S of size at most s, we can find a solution of LCP (M, q) in $n^{O(s)}$ time. In the following, we derive a necessary condition to reduce the number of enumerated index sets.

Note that, if $s \cdot \max(r, c) \geq n$, then the above simple exponential-time algorithm becomes a fixed-parameter one. We may henceforth assume that $s \cdot \max(r, c) < n$.

We denote $T = \{i \mid q_i < 0\}$. It is clear that if $T = \emptyset$, i.e., $q \geq 0$, then $z = 0$ is a solution of LCP (M, q). Thus we may assume that T is nonempty. Moreover, it is observed that for each $i \in T$, any solution z of LCP (M, q) has a positive entry z_j with $M_{ij} > 0$, since $M_{i.}z \geq -q_i > 0$. This implies that the size of T is not large as follows.

▶ **Lemma 2.** *For any solution z of* LCP (M, q), $|T| \leq c|S_z|$ *holds.*

We define an undirected graph $G = (V, E)$ by $V = [n]$ and $E = \{(i, j) \mid i \neq j$ and $(M_{ij} \neq 0$ or $M_{ji} \neq 0)\}$. For a vertex $i \in V$, let $N_+(i) = \{j \mid j \neq i,\ M_{ij} \neq 0\}$, and $N_-(j) = \{i \mid i \neq j,\ M_{ij} \neq 0\}$. For a vertex set V', we denote $N_-(V') = \bigcup_{j \in V'} N_-(j)$, $N(V') = \bigcup_{i \in V'}(N_+(i) \cup N_-(i))$, and $E_-(V') = \bigcup_{j \in V'}\{(i, j) \mid i \in N_-(j)\}$.

▶ **Definition 3.** A solution z of LCP (M, q) is *minimal* if there exists no solution z' of LCP (M, q) with $S_{z'} \subseteq S_z$.

A minimal solution has the following property.

▶ **Lemma 4.** *For any minimal solution z of* LCP (M, q), *each connected component of a subgraph $G' = (S_z \cup N_-(S_z), E_-(S_z))$ of G has a vertex in T.*

Proof. Let C be the vertex set of a connected component of G', and we show that $C \cap T$ is nonempty.

It is observed that $C \cap S_z$ is nonempty, since every edge in G' has an end vertex in S_z. Let $D = C \cap S_z$. By appropriately rearranging row and columns, the submatrix $M_{.S_z}$ of M is of the form

$$\begin{array}{l} \\ S_z \setminus C \\ C \\ N_-(S_z) \setminus C \\ V \setminus (S_z \cup N_-(S_z)) \end{array} \begin{array}{c} \begin{array}{cc} S_z \setminus C & D \end{array} \\ \begin{bmatrix} * & O \\ O & * \\ * & O \\ O & O \end{bmatrix} \end{array},$$

where $*$ denotes a matrix of appropriate size, and O is a zero matrix.

To show $C \cap T \neq \emptyset$, suppose to the contrary, that is, $q_C \geq 0$. We define a nonnegative vector z' by setting $z'_D = 0$ and $z'_{\overline{D}} = z_{\overline{D}}$. Then it holds that

$$(Mz' + q)_C = M_{CD}z'_D + q_C = q_C \geq 0,$$

and $(Mz' + q)_{\overline{C}} = (Mz + q)_{\overline{C}} \geq 0$. Thus $Mz' + q \geq 0$ holds. Moreover, since $z'_D = 0$ and $z'_{V \setminus S_z} = 0$, we have

$$(z')^\top(Mz' + q) = (z'_{S_z \setminus C})^\top(Mz' + q)_{S_z \setminus C} = (z_{S_z \setminus C})^\top(Mz + q)_{S_z \setminus C} = 0.$$

Thus z' is a solution of LCP (M, q). However it holds that $S_{z'} \subsetneq S_z$, which contradicts the minimality of z. Therefore, we have $q_C \not\geq 0$, and thus $C \cap T \neq \emptyset$. ◀

Lemma 4 suggests to enumerate all index sets S of size at most s such that

$$\begin{array}{l} \text{each connected component of } (S \cup N_-(S), E_-(S)) \text{ has a vertex in } T, \text{ and} \\ T \subseteq S \cup N_-(S). \end{array} \tag{3}$$

For the purpose, we first split T into two sets T_1 and T_2, and guess that $T_1 \subseteq S$ and $T_2 \subseteq N_-(S)$. Since $T_2 \subseteq N_-(S)$, each vertex in T_2 has to be connected to a vertex of S. We choose one index $p_i \in N_+(i)$ for each $i \in T_2$, and guess p_i's are contained in S. Thus at the beginning, $S_0 = T_1 \cup \{p_i \mid i \in T_2\}$ is supposed to be contained in S.

The algorithm traverses the graph G starting from $S = S_0$, and augment S keeping the condition (3). During the traversing procedure, each visited vertex is either active or inactive, where only active vertices can be chosen in the iteration.

Algorithm for the s-support (r, c)-sparse LCP

Input: A matrix M, a vector q and a positive integer s.

Step 0: Construct the undirected graph G as described before. Let $T = \{i \mid q_i < 0\}$, and let $\mathcal{S} = \emptyset$.
If $T = \emptyset$, return a solution $z = 0$. If $|T| > cs$ or $\{j \mid M_{ij} > 0\} = \emptyset$ for some $i \in T$, return that LCP (M, q) has no solution z with $|S_z| \leq s$.

Step 1: Set $\mathcal{T}$ to be the family of all partitions (T_1, T_2) of T. For each $(T_1, T_2) \in \mathcal{T}$, do Step 2.

Step 2: Set $\mathcal{P}$ to be the family of all index sets of size at most $|T_2|$ having one index $p_i \in N_+(i)$ for each $i \in T_2$. For each $P \in \mathcal{P}$, let $S_0 = T_1 \cup P$, and do Step 3.

Step 3: Set each vertex in $S_0 \cup T_2$ to be active.
While $|S| < s$ and there exists an active vertex, do the following steps.
- **(i)** take arbitrarily an active vertex i,
- **(ii)** if $i \notin S$, then either (a) set i to be inactive, or (b) visit a vertex $j \in N_+(i)$, make j active, and add j to S,
- **(iii)** if $i \in S$, then do exactly one of the following: (a) set i to be inactive, (b) visit a vertex $j \in N(i)$, make j active, and add j to S, or (c) visit a vertex $j \in N_-(i)$ and make j active,
- **(iv)** set $\mathcal{S} \leftarrow \mathcal{S} \cup \{S\}$ if S is updated.

Step 4: Check whether LCP (M, q) has a solution z with support S for some $S \in \mathcal{S}$. If exists, return the solution z. Otherwise, return that LCP (M, q) has no solution z with $|S_z| \leq s$.

The algorithm enumerates all the index sets satisfying (3) by the construction.

▶ **Claim 5.** *$\mathcal{S}$ contains all index sets satisfying* (3).

In the following, we show the running time of the algorithm. Note that for each vertex i of the graph G, we have $|N_+(i)| \leq r$ and $|N_-(i)| \leq c$ by (r, c)-sparsity.

It is not difficult to see that $|\mathcal{T}| = 2^{|T|}$ and $|\mathcal{P}| \leq r^{|T|}$. At any time in the algorithm, all active or inactive vertices are contained in $S \cup N_-(S)$. Thus in Step 3, there exist at most $(s + cs)$ active vertices, and, for each active vertex, there exist at most $(|N_+(i)| + |N_-(i)|) + |N_-(i)| \leq r + 2c$ choices of unvisited vertices in Step 3 (ii)-(iii). Thus each iteration has at most $(s + cs)(1 + r + 2c)$ branches. Moreover, the procedure is repeated at most $2(s + cs)$ times. Indeed, since the change of the state of vertices is irreversible, at most $s + cs$ vertices are set to be active, and some of them to be inactive. Therefore, the procedure generates at most $((s + cs)(1 + r + 2c))^{2(s+cs)}$ index sets. The existence of a solution of LCP (M, q) with support S can be decided in $n^{\mathrm{O}(1)}$ time. Since we have $|T| \leq cs$ by 2, the running time of the algorithm is bounded by

$$2^{cs} \cdot r^{cs} \cdot ((s + cs) \cdot (1 + r + 2c)))^{2(s+cs)} \cdot n^{\mathrm{O}(1)} \leq ((r + c)rcs)^{\mathrm{O}(cs)} \cdot n^{\mathrm{O}(1)}.$$

Then we have proved the following theorem.

▶ **Theorem 6.** *For any (r, c)-sparse LCP instance of order n, a solution with at most s nonzero entries can be found in $((r+c)rcs)^{\mathrm{O}(cs)} \cdot n^{\mathrm{O}(1)}$ time.*

3 Hardness results

In this section, we present hardness results for parameterized LCPs. First, we show the $W[1]$-hardness of the s-support r-row-sparse LCP with parameter $s + r$ by reducing the weighted 2SAT. Given a Boolean formula in conjunctive normal form ψ with at most two literals per clause and a positive integer s, the weighted 2SAT with parameter s is to decide whether there exists a satisfiable assignment in which at most s variables are set to be true. The problem is $W[1]$-complete with parameter s.

▶ **Theorem 7.** *The s-support 2-row-sparse LCP with parameter s is $W[1]$-hard.*

We also show the following theorem by reducing the s-subset sum problem with parameter s, which is $W[1]$-hard [14]. Given a set of integers $A = \{a_1, \ldots, a_n\}$, and integers t, s, the s-subset sum problem with parameter s is to decide whether there exists a set $S \subseteq [n]$ such that $|S| = s$ and $\sum_{i \in S} a_i = t$.

▶ **Theorem 8.** *The s-support 3-column-sparse LCP with parameter s is $W[1]$-hard.*

In the rest of this section, we show the NP-hardness of the $(2, 2)$-sparse LCP by reducing the *monotone one-in-three 3SAT*. Given a Boolean formula in conjunctive normal form $\psi = \bigwedge_{j=1}^{m}(x_{j_1} \vee x_{j_2} \vee x_{j_3})$ with three positive literals per clause, the monotone one-in-three 3SAT is to decide whether there exists an assignment such that each clause contains exactly one true literal.

This problem was introduced and proved to be NP-complete by Schaefer [26]. The monotone one-in-three 3SAT was also used to prove the NP-hardness of the 2-row-sparse LCP [27]. We obtain a stronger result by improving the proof with the following lemma.

▶ **Lemma 9.** *Let k be a positive integer, and consider an LCP instance*

$$\begin{aligned} &w_j = -z_{j+1} + 1 \geq 0, \qquad z_j \geq 0, \qquad w_j z_j = 0 \quad (j = 1, \ldots, k-1), \\ &w_k = -z_1 + 1 \geq 0, \qquad z_k \geq 0, \qquad w_k z_k = 0. \end{aligned} \tag{4}$$

The LCP instance (4) has only two solutions: $z = 0$ and $z = 1$.

▶ **Theorem 10.** *The $(2, 2)$-sparse LCP is NP-hard.*

Proof. Let $\psi = \bigwedge_{j=1}^{m}(x_{j_1} \vee x_{j_2} \vee x_{j_3})$ be a monotone one-in-three 3SAT instance with n literals. We construct an instance of the $(2, 2)$-sparse LCP of order $3mn + 6m$ from ψ as follows: for each literal $i = 1, \ldots, n$, letting $\alpha_i = 3m(i-1)$, define

$$\begin{aligned} w_{\alpha_i + j} &= -z_{\alpha_i + j + 1} + 1 \quad (j = 1, \ldots, 3m-1), \\ w_{\alpha_i + 3m} &= -z_{\alpha_i + 1} + 1. \end{aligned} \tag{5}$$

Moreover, for each clause $j = 1, \ldots, m$, letting $\beta_j = 3mn + 6(j-1)$ and $\gamma_\ell^j = \alpha_{j_\ell} + 3(j-1)$ $(\ell = 1, 2, 3)$, set

$$\begin{aligned} w_{\beta_j + 1} &= -z_{\gamma_2^j + 1} - z_{\gamma_3^j + 1} + 1, \\ w_{\beta_j + 2} &= -z_{\gamma_1^j + 2} - z_{\gamma_3^j + 2} + 1, \\ w_{\beta_j + 3} &= -z_{\gamma_1^j + 3} - z_{\gamma_2^j + 3} + 1, \end{aligned} \tag{6}$$

and in addition, set

$$\begin{aligned} w_{\beta_j+4} &= z_{\beta_j+1} + z_{\gamma_1^j+1} - 1, \\ w_{\beta_j+5} &= z_{\beta_j+2} + z_{\gamma_2^j+2} - 1, \\ w_{\beta_j+6} &= z_{\beta_j+3} + z_{\gamma_3^j+3} - 1. \end{aligned} \tag{7}$$

We denote by $w = Mz + q$ the system of linear equations consisting of the above constraints (5), (6), and (7). Consider LCP (M, q), i.e., $w = Mz + q \geq 0,\ z \geq 0,\ z^\top w = 0$. Since each constraint has at most two entries of z and each entry of z appears in at most two constraints, this is a $(2, 2)$-sparse LCP instance.

We will show that LCP(M, q) has a solution if and only if the monotone one-in-three 3SAT instance ψ is satisfiable.

First assume that LCP(M, q) has a solution (w, z). For each $i = 1, \ldots, n$, by applying Lemma 9 to (5), it holds that either

$$\begin{aligned} &w_{\alpha_i+1} = \cdots = w_{\alpha_i+3m} = 0 \text{ and } z_{\alpha_i+1} = \cdots = z_{\alpha_i+3m} = 1, \text{ or} \\ &w_{\alpha_i+1} = \cdots = w_{\alpha_i+3m} = 1 \text{ and } z_{\alpha_i+1} = \cdots = z_{\alpha_i+3m} = 0. \end{aligned} \tag{8}$$

Assign each literal x_i *true* if $z_{\alpha_i+1} = 1$, and *false* otherwise. We claim that x is a satisfiable assignment for ψ, that is, each clause has exactly one true literal. Indeed, for each clause j, if $z_{\gamma_1^j+1} = 0$, then $z_{\gamma_1^j+3} = 0$ by (8), and $z_{\beta_j+1} \geq 1 > 0$, $w_{\beta_j+1} = 0$ by (7) and the complementarity, and hence exactly one of $z_{\gamma_2^j+1}$ and $z_{\gamma_3^j+1}$ is equal to one by the first equation in (6). On the other hand, if $z_{\gamma_1^j+1} = 1$, then we have $z_{\gamma_1^j+2} = z_{\gamma_1^j+3} = 1$ by (8), and hence $z_{\gamma_2^j+3} = z_{\gamma_3^j+2} = 0$ by the second and third equations in (6). Thus $z_{\gamma_2^j+1} = z_{\gamma_3^j+1} = 0$, which means each clause has exactly one true literal.

Conversely, assume that $x = (x_1, \ldots, x_n)$ is a satisfiable assignment of ψ. Define $z \in \mathbb{R}^{3mn+6m}$ as follows: For $i = 1, \ldots, n$, set $z_{\alpha_i+1} = \cdots = z_{\alpha_i+3m} = 1$ if x_i is true, and $z_{\alpha_i+1} = \cdots = z_{\alpha_i+3m} = 0$ if x_i is false. For $j = 1, \ldots, m$, set $z_{\beta_j+\ell} = 1 - z_{\gamma_\ell^j+\ell}$ for $\ell = 1, 2, 3$ and $z_{\beta_j+\ell} = 0$ for $\ell = 4, 5, 6$. Define w to be $w = Mz + q$. Then the pair (w, z) satisfies (5), (6), and (7), as (6) follows from that x is a satisfiable assignment. Moreover, $w, z \geq 0$ holds.

In order to prove that z is a solution of LCP(M, q), it remains to show that the pair (w, z) satisfies $w^\top z = 0$. For $i = 1, \ldots, n$, it holds that $w_{\alpha_i+\ell} z_{\alpha_i+\ell} = 0$ $(\ell = 1, \ldots, 3m)$ since (8) is satisfied. Let $j \in \{1, \ldots, m\}$. Since the clause j has exactly one true literal, we may suppose by symmetry that $z_{\gamma_1^j+1} = 1$ and $z_{\gamma_2^j+1} = z_{\gamma_3^j+1} = 0$. By (6), it holds that $w_{\beta_j+1} = 1$ and $w_{\beta_j+2} = w_{\beta_j+3} = 0$. On the other hand, we have $z_{\beta_j+1} = 1 - z_{\gamma_1^j+1} = 0$ by definition, which means that $w_{\beta_j+\ell} z_{\beta_j+\ell} = 0$ for $\ell = 1, 2, 3$. For $\ell = 4, 5, 6$, we have $w_{\beta_j+\ell} z_{\beta_j+\ell} = 0$, since $z_{\beta_j+\ell} = 0$. Thus the complementarity condition is satisfied. Therefore, LCP(M, q) has a solution if and only if ψ is satisfiable, which completes the proof. ◀

4 Polynomial kernel

In this section, we discuss the existence of a polynomial kernel for the parameterized LCP. For a matrix M and a vector q, let size(M) and size(q) denote the data size of M and q, respectively. For the s-support (r, c)-sparse LCP with parameter $s + r + c$, a polynomial kernelization algorithm is that given an (r, c)-sparse LCP instance LCP (M, q) and an integer s, outputs in poly(size(M) + size(q) + $s)$ time an (r', c')-sparse LCP instance LCP (M', q') and an integer s' such that

- LCP (M, q) has a solution with support size at most s
  $\Leftrightarrow$ LCP (M', q') has a solution with support size at most s',
- $\mathrm{size}(M') + \mathrm{size}(q') + r' + c' + s' \leq \mathrm{poly}(r + c + s)$.

We show the following theorem.

▶ **Theorem 11.** *The s-support (r, c)-sparse LCP with parameter $s + r + c$ has no polynomial kernel unless* coNP $\subseteq$ NP/poly.

Bodlaender, Downey, Fellows and Hermelin [3] proposed a general framework to prove the nonexistence of a polynomial kernel. They showed that, for any parameterized problem whose unparameterized version is NP-complete, if it admits an algorithm called a *composition algorithm*, then it has no polynomial kernel unless coNP $\subseteq$ NP/poly. We prove Theorem 11 by using their framework.

Note that the unparameterized version of our problem is the s-support LCP, and the decision version is NP-complete by reduction from the s-support bimatrix game, which is NP-complete [18]. For a square matrix M, we denote by $r(M)$ and $c(M)$ the maximum numbers of nonzero entries per row and column in M, respectively.

▶ **Definition 12.** A *composition algorithm* for the s-support (r, c)-sparse LCP with parameter $s + r + c$ is that given t pairs of an LCP instance LCP $\left(M^i, q^i\right)$ and an integer s_i $(i = 1, \ldots, t)$, where $s_i + r(M^i) + c(M^i) = k$ for all i, outputs in $\mathrm{poly}(\sum_{i=1}^{t}(\mathrm{size}(M^i) + \mathrm{size}(q^i)) + k)$ time an (r', c')-sparse LCP instance LCP (M', q') and an integer s' such that

- LCP (M', q') has a solution with support size at most s'
  $\Leftrightarrow$ some LCP $\left(M^i, q^i\right)$ has a solution with support size at most s_i,
- $s' + r' + c' \leq \mathrm{poly}(k)$.

In the following, we construct a composition algorithm for the s-support LCP.

▶ **Lemma 13.** *The s-support (r, c)-LCP has a composition algorithm.*

Proof. Suppose that we are given t pairs of an LCP instance LCP $\left(M^i, q^i\right)$ and an integer s_i $(i = 1, \ldots, t)$, where $s_i + r(M^i) + c(M^i) = k$ for all i. Let $s = \max_i s_i$, $r = \max_i r(M^i)$, and $c = \max_i c(M^i)$. First, for each $i = 1, \ldots, t$, we define an LCP instance as

$$\begin{bmatrix} M^i & O \\ O & I_{s-s_i} \end{bmatrix} \begin{pmatrix} z \\ x \end{pmatrix} + \begin{bmatrix} q^i \\ -1 \end{bmatrix},$$

where I_{s-s_i} is the identity matrix whose size is $s - s_i$, and O is a zero matrix. The coefficient matrix and the constant vector are denoted by N^i and d^i, respectively. Since any solution of LCP $\left(N^i, d^i\right)$ satisfies $x = 1$, it is not difficult to see the following claim.

▶ **Claim 14.** *For each $i = 1, \ldots, t$, it holds that* LCP $\left(M^i, q^i\right)$ *has a solution with support size at most s_i if and only if* LCP $\left(N^i, d^i\right)$ *has a solution with support size at most s.*

We remark that $\mathrm{size}(N^i)$ is bounded by $\mathrm{poly}(s + \mathrm{size}(M^i))$, and $\mathrm{size}(d^i)$ is bounded by $\mathrm{poly}(s) + \mathrm{size}(q^i)$. Since $r(N^i) \leq r$ and $c(N^i) \leq c$ for each i, every instance LCP $\left(N^i, d^i\right)$ is (r, c)-sparse.

The output (M', q', s') of our composition algorithm is constructed as follows. Let $T_i = \{j \mid d^i_j < 0\}$ for $i = 1, \ldots, t$. If $|T_i| > cs$, then Lemma 2 implies that LCP $\left(N^i, d^i\right)$ does not have a solution with support size at most s. Thus by removing such instances, we may assume that $|T_i| \leq cs$ for all $i = 1, \ldots, t$. For each $i = 1, \ldots, t$, let ℓ^i be a vector whose j-th

entry is $\min(0, d^i_j)$, and let $u^i = d^i - \ell^i \geq 0$. We define a matrix M', a vector q' and an integer s' as follows:

$$M' = \left[\begin{array}{ccccc|c} N^1 & \ell^1 & & & & \\ 0 & -1 & & \multicolumn{2}{c|}{\multirow{2}{*}{O}} & \\ & & \ddots & & & 0 \\ & & & N^t & \ell^t & \\ \multicolumn{2}{c}{\multirow{2}{*}{O}} & & 0 & -1 & \\ & & & & & \\ \hline 0 & 1 & \cdots & 0 & 1 & 0 \end{array}\right], \quad q' = \left[\begin{array}{c} u^1 \\ 1 \\ \vdots \\ u^t \\ 1 \\ \hline -1 \end{array}\right], \text{ and } s' = s + 1.$$

We can construct M', q', s' in time

$$\text{poly}\left(\sum_{i=1}^{t}(\text{size}(N^i) + \text{size}(d^i)) + (s + r + c)\right) \leq \text{poly}\left(\sum_{i=1}^{t}(\text{size}(M^i) + \text{size}(q^i)) + k\right).$$

We first claim that the output instance satisfies the second condition of the definition of a composition algorithm. It is observed that M' has at most $r + 1 \leq k + 1$ nonzero entries per row, and $\max_i |T^i| + 2 \leq cs + 2 \leq k^2 + 2$ nonzero entries per column. Thus the parameter $s' + r(M') + c(M')$ of LCP (M', q') is bounded by $(s + 1) + (k + 1) + (k^2 + 2) \leq \text{poly}(k)$.

The rest of the proof shows that the first condition of the definition is satisfied, which completes the proof of the lemma.

▶ **Claim 15.** LCP (M', q') *has a solution with support size at most* s' *if and only if* LCP $\left(N^i, d^i\right)$ *has a solution with support size at most* s *for some* $1 \leq i \leq t$.

◀

Acknowledgements. The first author is supported by Grant-in-Aid for JSPS Fellows, and by JST, ERATO, Kawarabayashi Large Graph Project. The second author is supported by JSPS KAKENHI Grant Numbers 25730001 and 24106002, and by JST, ERATO, Kawarabayashi Large Graph Project. The third author is supported by JSPS KAKENHI Grant Numbers 24106002 and 26280001.

References

1 V. Arvind, J. Köbler, S. Kuhnert, and J.Torán. Solving linear equations parameterized by Hamming weight. In *Proceedings of the 9th International Symposium on Parameterized and Exact Computation*, pages 39–50, 2014.

2 H. Björklund, O. Svensson, and S. Vorobyov. Linear complementarity algorithms for mean payoff games. Technical Report 2005-05, DIMACS: Center for Discrete Mathematics and Theoretical Computer Science, 2005.

3 H. L. Bodlaender, R. G. Downey, M. R. Fellows, and D. Hermelin. On problems without polynomial kernels. *Journal of Computer and System Sciences*, 75:423–434, 2009.

4 X. Chen, X. Deng, and S. Teng. Sparse games are hard. In *Proceedings of the 2nd International Workshop on Internet and Network Economics*, pages 262–273, 2006.

5 X. Chen, X. Deng, and S. Teng. Settling the complexity of computing two-player Nash equilibria. *Journal of the ACM*, 56:14:1–14:57, 2009.

6 S. J. Chung. NP-completeness of the linear complementarity problem. *Journal of Optimization Theory and Applications*, 60:393–399, 1989.

7 B. Codenotti, M. Leoncini, and G. Resta. Efficient computation of Nash equilibria for very sparse win-lose bimatrix games. In *Proceedings of the 14th Annual European Symposium on Algorithms*, pages 232–243, 2006.

8 E. Cohen and N. Megiddo. Improved algorithms for linear inequalities with two variables per inequality. *SIAM Journal on Computing*, 23:1313–1347, 1994.

9 R. W. Cottle. The principal pivoting method of quadratic programming. In *Mathematics of Decision Sciences*, Part 1, pages 142–162. American Mathematical Society, Providence R. I., 1968.

10 R. W. Cottle and G. B. Dantzig. Complementary pivot theory of mathematical programming. *Linear Algebra and Its Applications*, 1:103–125, 1968.

11 R. W. Cottle, J. S. Pang, and R. E. Stone. *The Linear Complementarity Problem*. Academic Press, Boston, 1992.

12 P. Damaschke. Sparse solutions of sparse linear systems: fixed-parameter tractability and an application of complex group testing. *Theoretical Computer Science*, 511:137–146, 2013.

13 C. Daskalakis and C. H. Papadimitriou. On oblivious PTAS's for Nash equilibrium. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, pages 75–84, 2009.

14 R. G. Downey and M. R. Fellows. Fixed-parameter intractability. In *Proceedings of the 7th Annual Structure in Complexity Theory Conference*, pages 36–49, 1992.

15 R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, New York, 1999.

16 V. Estivill-Castro and M. Parsa. Computing Nash equilibria gets harder: new results show hardness even for parameterized complexity. In *Proceedings of the 15th Australasian Symposium on Computing*, pages 83–90, 2009.

17 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, Berlin, 2006.

18 I. Gilboa and E. Zemel. Nash and correlated equilibria: some complexity considerations. *Games and Economic Behavior*, 1:80–93, 1989.

19 D. Hermelin, C. Huang, S. Kratsch, and M. Wahlström. Parameterized two-player Nash equilibrium. *Algorithmica*, 65:1–15, 2013.

20 D. S. Hochbaum and J. Naor. Simple and fast algorithms for linear and integer programs with two variables per inequality. *SIAM Journal on Computing*, 23:1179–1192, 1994.

21 S. Kratsch. On polynomial kernels for integer linear programs: covering, packing and feasibility. In *Proceedings of the 21st Annual European Symposium*, pages 647–658, 2013.

22 S. Kratsch. On polynomial kernels for sparse integer linear programs. In *Proceedings of the 30th Symposium on Theoretical Aspects of Computer Science*, pages 80–91, 2013.

23 S. Kratsch and V. A. Quyen. On kernels for covering and packing ILPs with small coefficients. In *Proceedings of the 9th International Symposium on Parameterized and Exact Computation*, pages 307–318, 2014.

24 C. E. Lemke. Bimatrix equilibrium points and mathematical programming. *Management Science*, 11:681–689, 1965.

25 K. G. Murty. *Linear Complementarity, Linear and Nonlinear Programming*. Internet Edition, 1997.

26 T. J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing*, pages 216–226, 1978.

27 H. Sumita, N. Kakimura, and K. Makino. The linear complementarity problems with a few variables per constraint. *Mathematics of Operations Research*, to appear.

Parameterized Lower Bound and Improved Kernel for Diamond-free Edge Deletion

R. B. Sandeep and Naveen Sivadasan

Department of Computer Science & Engineering
Indian Institute of Technology Hyderabad, India*
{cs12p0001,nsivadasan}@iith.ac.in

Abstract

A diamond is a graph obtained by removing an edge from a complete graph on four vertices. A graph is diamond-free if it does not contain an induced diamond. The Diamond-free Edge Deletion problem asks to find whether there exist at most k edges in the input graph whose deletion results in a diamond-free graph. The problem was proved to be NP-complete and a polynomial kernel of $O(k^4)$ vertices was found by Fellows et. al. (Discrete Optimization, 2011).

In this paper, we give an improved kernel of $O(k^3)$ vertices for Diamond-free Edge Deletion. We give an alternative proof of the NP-completeness of the problem and observe that it cannot be solved in time $2^{o(k)} \cdot n^{O(1)}$, unless Exponential Time Hypothesis fails.

1998 ACM Subject Classification F.2.2. Nonnumerical Algorithms and Problems

Keywords and phrases edge deletion problems, polynomial kernelization

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.365

1 Introduction

For a finite set of graphs $\mathcal{H}$, $\mathcal{H}$-free Edge Deletion problem asks whether we can delete at most k edges from an input graph G to obtain a graph G' such that for every $H \in \mathcal{H}$, G' does not have an induced copy of H. If $\mathcal{H} = \{H\}$, the problem is denoted by H-free Edge Deletion. $\mathcal{H}$-free Edge Deletion comes under the broader category of graph modification problems which have found applications in DNA physical mapping [3, 13], circuit design [9] and machine learning [2]. Cai has proved that $\mathcal{H}$-free Edge Deletion is fixed parameter tractable [4]. Polynomial kernelization and incompressibility of these problems were subjected to rigorous studies in the recent past. Kratsch and Wahlström gave the first example on the incompressibility of H-free Edge Deletion problems by proving that the problem is incompressible if H is a certain graph on seven vertices, unless NP $\subseteq$ coNP/poly [15]. Later, it has been proved that there exist no polynomial kernel for H-free Edge Deletion where H is any 3-connected graph other than a complete graph, unless NP $\subseteq$ coNP/poly [5]. In the same paper, under the same assumption, it is proved that, if H is a path or a cycle, then H-free Edge Deletion is incompressible if and only if H has at least four edges. It has been proved that $\mathcal{H}$-free Edge Deletion admits polynomial kernelization on bounded degree graphs if $\mathcal{H}$ is a finite set of connected graphs [1]. Though polynomial kernels have been found for many H-free Edge Deletion problems, Claw-free Edge Deletion withstood the test of time and yielded neither an incompressibiltiy result nor a polynomial kernel. Some progress has been made recently for

* The first author is supported by TCS Research Scholarship.

this problem such as a polynomial kernel for Claw-free Edge Deletion on K_t-free input graphs [1] and a polynomial kernel for {Claw, Diamond}-free Edge Deletion [7].

In this paper, we study the polynomial kernelization and parameterized lower bound of Diamond-free Edge Deletion. It has been proved that Diamond-free Edge Deletion is NP-complete and admits a kernel of $O(k^4)$ vertices [11][1]. We improve this result by giving a kernel of $O(k^3)$ vertices. We use vertex modulator technique, which was used recently to give a polynomial kernel for Trivially Perfect Editing [8] and to obtain a polynomial kernel for {Claw, Diamond}-free Edge Deletion [7]. We introduce a rule named as *Vertex Split* which *splits* a vertex into a set of independent vertices where each vertex in the set corresponds to a component in the neighborhood of the vertex. We believe that this rule may have further applications in similar settings.

We give an alternative proof of the NP-completeness of Diamond-free Edge Deletion. Our reduction is from the Vertex Cover problem on cubic graphs and is a linear parameterized reduction. This enables us to prove that, unless Exponential Time Hypothesis (ETH) fails, there exists no parameterized subexponential time algorithm (an algorithm which runs in time $2^{o(k)} \cdot n^{O(1)}$) for Diamond-free Edge Deletion.

1.1 Preliminaries

The problem we consider in this paper is Diamond-free Edge Deletion: whether there exist at most k edges whose deletion from the input graph results in a graph without any induced diamonds. In the parameterized version, the parameter is k.

Graphs: Every graph considered here is simple, finite and undirected. For a graph G, $V(G)$ and $E(G)$ denote the vertex set and the edge set of G respectively. $N_G(v)$ denotes the (open) neighborhood of a vertex $v \in V(G)$, which is the set of vertices adjacent to v in G. The closed neighborhood of v is denoted by $N_G[v]$ and is defined by $N_G(v) \cup \{v\}$. We remove the subscript when there is no ambiguity about the underlying graph G. A graph $G' = (V', E')$ is called an induced subgraph of a graph G if $V' \subseteq V(G)$, $E' \subseteq E(G)$ and an edge $\{x, y\} \in E(G)$ is in E' if and only if $\{x, y\} \subseteq V'$. For a vertex set $V' \subseteq V(G)$, $G[V']$ denotes the induced subgraph with a vertex set V' of G. A component G' of a graph G is a connected induced subgraph of G such that there is no edge between $V(G')$ and $V(G) \setminus V(G')$. For a set of vertices $V' \subseteq V(G)$, $G - V'$ denotes the graph obtained by removing the vertices in V' and all its incident edges from G. For an edge set $E' \subseteq E(G)$, $G - E'$ denotes the graph obtained by deleting all edges in E' from G. If V' (E') is a singleton set $\{v\}$ ($\{e\}$), we denote the graph $G - V'$ ($G - E'$) by $G - v$ ($G - e$). For an edge set $E' \subseteq E(G)$, $V_{E'}(G)$ denotes the vertices in G incident to the edges in E'. A matching (non-matching) is a set of edges (non-edges) such that every vertex in the graph is incident to at most one edge (non-edge) in the matching (non-matching). Diamond is a graph obtained by deleting an edge from a complete graph on four vertices. A graph G is called diamond-free, if G does not contain any diamond as an induced subgraph. Whenever we mention that $\{a, b, c, d\} \subseteq V(G)$ induces a diamond in G, a and b are degree-3 vertices and c and d are degree-2 vertices. In a diamond, we call the edge between the degree-3 vertices as the *middle* edge.

[1] We came to know about this result only after the acceptance of this paper. During this work, our reference was a kernel of $O(k^5)$ vertices [6].

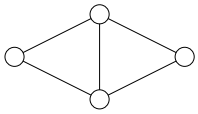


Figure 1 Diamond.

Parameterized complexity: A parameterized problem is *fixed parameter tractable*, if there is an algorithm to solve it in time $f(k) \cdot n^{O(1)}$, where f is any computable function and n is the size of the input, and k is the parameter. A *Polynomial kernelization* is an algorithm which takes as input (G, k), an instance of a parameterized problem, runs in time $(|G|+k)^{O(1)}$ and returns an instance (G', k') of the same problem such that $|G'|, k' \leq p(k)$, where p is any polynomial function. A rule for kernelization is *safe* if (G, k) is a yes-instance if and only if (G', k') is a yes-instance where (G, k) and (G', k') are the input and output of the kernelization. *Linear parameterized reduction* from a parameterized problem A to another B is a polynomial time reduction such that $k' = O(k)$ where k and k' are the parameters of the instances of A and B respectively. A *subexponential time algorithm* for a parameterized problem is an algorithm which runs in time $2^{o(k)} \cdot n^{O(1)}$ where n is the size of the problem instance.

2 Polynomial Kernel

In this section, we give a kernel with $O(k^3)$ vertices for Diamond-free Edge Deletion. Due to space constraints, some of the proofs are deferred to the full version of this paper. To start with, we introduce two properties of graphs and two rules based on those properties.

▶ **Definition 1** (Core Member)**.** A vertex or an edge of a graph G is a *core member* of G if it is a part of some induced diamond or K_4 in G. G has core member property if every vertex and every edge of G is a core member.

▶ **Rule 1** (Irrelevant Edge)**.** *Let (G, k) be an input to the rule. If there is an edge $e \in E(G)$ which is not a core member of G, then delete e from G.*

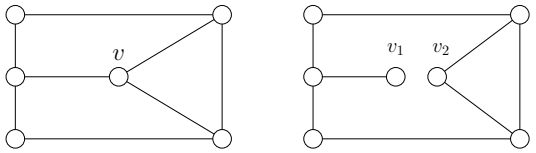
▶ **Lemma 2.** *Irrelevant edge rule is safe and can be applied in polynomial time.*

Proof Idea. An edge which is not a core member is not a part of any minimum solution. Deleting such an edge will not create new diamonds. ◀

▶ **Definition 3** (Connected Neighborhood)**.** For a graph G and a vertex $v \in V(G)$, v has *connected neighborhood* if $G[N(v)]$ is connected. G has connected neighborhood if every vertex in G has connected neighborhood.

▶ **Rule 2** (Vertex-Split)**.** *Let $v \in V(G)$ and v does not have connected neighborhood in G. Let there be $t > 1$ components in $G[N(v)]$ with vertex sets $V_1, V_2, \ldots, V_t$. Introduce t new vertices $v_1, v_2, \ldots, v_t$ and make v_i adjacent to all vertices in V_i for $1 \leq i \leq t$. Delete v.*

An example of the application of vertex-split rule is depicted in Figure 2. We denote the set of vertices created by splitting v by V_v. Let G' be the graph obtained by splitting a vertex v in G. For convenience, we identify an edge (v, u) in G with an edge (v_j, u) in G' where u is in the j^{th} component of $G[N(v)]$, so that for every set of edges S in G, there is a corresponding set of edges in G' and vice versa. We identify a set of vertices $V' \subseteq V(G) \setminus \{v\}$ with the corresponding vertices in G'. Similarly, we identify $V' \subseteq V(G') \setminus V_v$ with the



(a) A graph G where a vertex v is eligible for a split.

(b) Vertex-split rule is applied at v.

Figure 2 An application of vertex-split rule.

corresponding vertex set in G. Before proving the safety of the rule, we prove two simple observations.

▶ **Observation 4.** *Let vertex-split rule be applied on G to obtain G'. Let $v \in V(G)$ be the vertex being split. Then:*

(i) *For every pair of vertices $\{v_i, v_j\} \subseteq V_v$, the distance between v_i and v_j is at least four.*

(ii) *Let $u \in V(G) \setminus \{v\}$ and u has connected neighborhood in G. Then u has connected neighborhood in G'. Furthermore, every new vertex v_i introduced in G' has connected neighborhood.*

Proof. (i). Let $\{v_i, v_j\} \subseteq V_v$. Clearly, v_i and v_j are non-adjacent. Consider any two vertices $u_i \in N(v_i)$ and $u_j \in N(v_j)$. If $u_i = u_j$ or u_i and u_j are adjacent in G', there would be only one vertex generated for the component containing u_i and u_j in $G[N(v)]$ by splitting v, which is a contradiction. It follows that the distance between v_i and v_j is at least four.

(ii). If $v \notin N_G(u)$, then the neighborhood of u is same in both G and G' and hence u has connected neighborhood in G'. Let $v \in N(u)$. Since $G[N_G(u)]$ is connected, to prove that $G'[N_{G'}(u)]$ is connected, it is enough to get an isomorphism between $G[N_G(u)]$ and $G'[N_{G'}(u)]$. Let V' be the set of all vertices in $N_G[u]$ to which v is adjacent. We note that $u \in V'$. Let v_i be the vertex generated by splitting v for the component in $G[N(v)]$ containing u. Since, there is only one new vertex introduced for a component of $G[N(v)]$, no other new vertex is adjacent to u in G'. Now, let V'' be the set of all vertices in $N_{G'}[u]$ to which v_i is adjacent to. Proving $V' = V''$ will establish an isomorphism between $G[N_G(u)]$ and $G'[N_{G'}(u)]$. In order to prove that $V' = V''$ it is enough to prove that $G[V']$ is connected. This is true since $u \in V'$ and u is adjacent to all other vertices in V'. Since a new vertex is made adjacent to a component in the neighborhood of v, every new vertex v_j in G' has connected neighborhood. ◀

▶ **Lemma 5.** *Vertex-split rule is safe and can be applied in polynomial time.*

Proof Idea. Splitting a vertex neither creates nor introduces a new diamond. It does not affect the propagation of the diamonds due to deleting edges. ◀

The next rule deletes an edge e, if e is the middle edge of $k+1$ otherwise edge-disjoint diamonds. This rule is found in [6].

▶ **Rule 3** (Sunflower)**.** *Let (G, k) be an input to the rule. If there is an edge $e = \{x, y\} \in E(G)$ such that $G[N(x) \cap N(y)]$ has a non-matching of size at least $k+1$, then delete e from G and decrease k by 1.*

▶ **Lemma 6.** *Sunflower rule is safe and can be applied in polynomial time.*

The next rule is a trivial one.

▶ **Rule 4** (Irrelevant component). *Let (G, k) be an input to the rule. If a component of G is diamond-free, then delete the component from G.*

▶ **Lemma 7.** *Irrelevant component rule is safe and can be applied in polynomial time.*

Now, we are ready with the Phase 1 of the kernelization.

Phase 1

Let (G, k) be an input to Phase 1.

- Exhaustively apply rules irrelevant edge, vertex split, sunflower and irrelevant component on (G, k) to obtain (G', k).

▶ **Lemma 8.** *Let (G', k') be obtained by applying Phase 1 on (G, k). Then:*

(i) *G' has core member property.*
(ii) *G' has connected neighborhood.*
(iii) *Every component in G' has an induced diamond.*
(iv) *$|E(G')| \leq |E(G)|$ and $|V(G')| \leq 2|E(G)|$.*

Proof. (i), (ii) and (iii) follow from the fact that irrelevant edge, vertex-split and irrelevant component rules are not applicable on (G', k). (iv) follows from the fact that none of the rules increases the number of edges in the graph. ◀

▶ **Lemma 9.** *Applying Phase 1 is safe and Phase 1 runs in polynomial time.*

Proof Idea. Follows from Lemma 8(iv) and the safety and running time of each rule. ◀

We define a vertex modulator for Diamond-free Edge Deletion similar to that defined for Trivially Perfect Editing [8].

▶ **Definition 10** (D-modulator). Let (G, k) be an instance of Diamond-free Edge Deletion. Let $V' \subseteq V(G)$ be such that $G[V \setminus V']$ is diamond-free. Then, V' is called a D-modulator.

▶ **Lemma 11** ([10]). *A graph G is diamond-free if and only if every edge in G is a part of exactly one maximal clique.*

For a diamond-free graph G, since every edge is in exactly one maximal clique, there is a unique way of partitioning the edges into maximal cliques. For convenience, we call the set of subsets of vertices, where each subset is the vertex set of a maximal clique, as a *maximal clique partitioning*. We note that, one vertex may be a part of many sets in the partitioning.

▶ **Lemma 12.** *Let (G, k) be an instance of* Diamond-free Edge Deletion. *Then, in polynomial time, the edge set X of a maximal set of edge-disjoint diamonds, a D-modulator V_X of size at most $4k$ and a maximal clique partitioning $\mathcal{C}$ of $G[V(G) \setminus V_X]$ can be obtained or it can be declared that (G, k) is a no-instance.*

Proof Idea. V_X is a set of vertices incident to X. $\mathcal{C}$ can be computed by a greedy method. ◀

Let (G, k) be an output of Phase 1. Here onward, we assume that X is an edge set of the maximal set of edge-disjoint diamonds, V_X is a D-modulator, which is the set of vertices incident to X and $\mathcal{C}$ is the maximal clique partitioning of $G[V(G) \setminus V_X]$. Observation 13 directly follows from the maximality of X. Observation 14 is found in Lemma 3.1 of [7]. It was proved there, if G is {claw, diamond}-free, but is also applicable if G is diamond-free.

▶ **Observation 13.** *Every induced diamond in G has an edge $\{a, b\}$ such that $\{a, b\} \in X$.*

▶ **Observation 14.** *Let $C, C' \in \mathcal{C}$ and be distinct. Then:*
(i) $|C \cap C'| \leq 1$.
(ii) *If $v \in C \cap C'$, then there is no edge between $C \setminus \{v\}$ and $C' \setminus \{v\}$.*

Proof. (i). Assume that $x, y \in C \cap C'$. Then the edge $\{x, y\}$ is part of two maximal cliques, which is a contradiction by Lemma 11.

(ii). Let $x \in C \setminus \{v\}$ and $y \in C' \setminus \{v\}$. Let x and y be adjacent. Clearly, $\{x, y\}$ is not part of the clique induced by C. Now, $\{x, v\}$ is part of not only the clique induced by C but also a maximal clique containing x, y and v, which is a contradiction. ◀

▶ **Definition 15** (Local Vertex)**.** Let G be a graph and $C \subseteq V(G)$ induces a clique in G. A vertex v in C is called local to C in G, if $N(v) \subseteq C$.

▶ **Lemma 16.** *Let (G, k) be an instance of* Diamond-free Edge Deletion*. Let C be a clique with at least $2k + 2$ vertices in G.*
(i) *Every solution S of size at most k of (G, k) does not contain any edge e where both the end points of e are in C.*
(ii) *Let $C' \subseteq C$ be such that every vertex $v \in C'$ is local to C in G. Every induced diamond with vertex set D in G can contain at most one vertex in C'.*
(iii) *Let $C' \subseteq C$ be such that every vertex $v \in C'$ is local to C in G. Then, it is safe to delete $\min\{|C'| - 1, |C| - (2k + 2)\}$ vertices of C' in G.*

Proof Idea. (i). Deleting an edge from a large clique will introduce unmanageable number of diamonds. (ii) follows from the properties of local vertices. (iii). In a big clique, retaining a single local vertex and deleting all other local vertices is safe. ◀

We partition $\mathcal{C}$ into three - $\mathcal{C}_1, \mathcal{C}_2$ and $\mathcal{C}_{\geq 3}$, the sets of vertices of maximal cliques with one, two and three or more vertices respectively. The first in the following observation has been proved in Lemma 3.2 in [7] in the context where $G - V_X$ is {diamond,claw}-free. Here we prove it in the context where $G - V_X$ is diamond-free.

▶ **Observation 17.** *Let $C \in \mathcal{C}$. Then:*
(i) *If there is a vertex $v \in V_X$ such that v is adjacent to at least two vertices in C, then v is adjacent to all vertices in C.*
(ii) *A vertex in $V(G) \setminus (V_X \cup C)$ is adjacent to at most one vertex in C.*

Proof. (i). Let v is adjacent to two vertices in x, y in C but not adjacent to $z \in C$. Then $\{x, y, v, z\}$ induces a diamond such that none of the edges of the diamond is in X.

(ii). Assume that a vertex $u \in V(G) \setminus (V_X \cup C)$ is adjacent to all vertices in C. This contradicts with the fact that C induces a maximal clique in $G - V_X$. Let u be adjacent to at least two vertices $\{a, b\}$ in C and non-adjacent to at least one vertex $v \in C$. Then $\{a, b, u, v\}$ induces a diamond where none of the edges of the diamond is in X. ◀

Consider $C \in \mathcal{C}$. We define three sets of vertices in G based on C.

$$\begin{aligned} A_C &= \{v \in V_X : v \text{ is adjacent to all vertices in } C\} \\ B_C &= \{v \in V(G) \setminus (V_X \cup C) : v \text{ is adjacent to exactly one vertex in } C\} \\ D_C &= \{v \in V_X : v \text{ is adjacent to exactly one vertex in } C\} \end{aligned}$$

For a vertex $v \in C$, let B_v denote the set of all vertices in B_C adjacent to v. Similarly let D_v denote the set of all vertices in D_C adjacent to v.

▶ **Observation 18.** *Let $C \in \mathcal{C}$. Then,*
(i) *The set of vertices in $V(G) \setminus C$ adjacent to at least one vertex in C is $A_C \cup B_C \cup D_C$.*
(ii) *If $|C| > 1$, then A_C induces a clique in G.*
(iii) *For two vertices $u, v \in C$, $B_u \cap B_v = \emptyset$ and $D_u \cap D_v = \emptyset$.*

Proof. (i) directly follows from Observation 17.

(ii). Assume not. Let a and b be two non-adjacent vertices in A_C. By Observation 17(i), both a and b are adjacent to all vertices in C. Consider any two vertices $x, y \in C$. $\{x, y, a, b\}$ induces a diamond with no edge in X, which is a contradiction.

(iii) directly follows from the definition of B_C and D_C. ◀

▶ **Lemma 19.** *Let $v \in C \in \mathcal{C}_{\geq 3}$. If B_v is non-empty then D_v is non-empty.*

Proof. Since v has connected neighborhood, $G[N(v)]$ is connected. We observe that $N(v) = A_C \cup B_v \cup D_v \cup (C \setminus \{v\})$. Assume B_v is non-empty. There is no edge between the sets B_v and $C \setminus \{v\}$. Consider a vertex $v_b \in B_v$ adjacent to $A_C \cup D_v$. Assume v_b is not adjacent to D_v. Then v_b must be adjacent to a vertex $v_a \in A_C$. Let v' be any other vertex in C. Then $\{v_a, v, v', v_b\}$ induces a diamond which has no edge intersection with X. Therefore v_b must be adjacent to a vertex in D_v. ◀

▶ **Observation 20.** *Let $C \in \mathcal{C}$. Then there are two adjacent vertices x and y such that $x \in A_C$ and $y \in A_C \cup D_C$.*

Proof.
Case 1: $C = \{v\} \in \mathcal{C}_1$. Since $\{v\} \in \mathcal{C}_1$, v is not adjacent to any vertex in $V(G) \setminus V_X$. Since v is a core member, v is part of an induced diamond or K_4 in G. Hence there exist two adjacent vertices $x, y \in A_C$.

Case 2: $C = \{u, v\} \in \mathcal{C}_2$. Since the edge $\{u, v\}$ is a core member, it is part of some induced diamond or K_4 in G. Let a, b be the other two vertices in an induced diamond or K_4 in which $\{u, v\}$ is a part. If both $a, b \in V(G) \setminus V_X$, then it contradicts with either the maximality of X (if a, b, u and v induce a diamond) or with the fact that $\{u, v\}$ is part of exactly one maximal clique C (if a, b, u and v induce a K_4). Let $a \in V_X$ and $b \in V(G) \setminus V_X$. Then, if a, b, u and v induces a diamond, then it contradicts with the maximality of X. If a, b, u and v induces a K_4, then u, v and b induce a K_3 which contradicts with the fact that $\{u, v\}$ is a part of exactly one maximal clique. Hence $a, b \in V_X$. Since a, b, u, v induce a diamond or a K_4, one of a, b must be adjacent to both u and v and the other vertex must be adjacent to at least one of u and v.

Case 3: $C \in \mathcal{C}_{\geq 3}$. Assume that $|A_C| = 0$. If $B_C \cup D_C = \emptyset$, then by Observation 18(i), the clique C is a component in G. Then, irrelevant component rule is applicable. Hence $B_C \cup D_C$ is non-empty. Consider a vertex $v \in C$ such that $B_v \cup D_v$ is non-empty. Consider $N(v)$. $G[N(v)]$ has at least two components, one from $B_v \cup D_v$ and the other from C, which contradicts with the fact that v has connected neighborhood. Hence, $|A_C| > 0$. Assume $|A_C = \{x\}| = 1$. For a contradiction, assume that $D_C = \emptyset$. Then Lemma 19 implies that B_C is empty. Then x does not have connected neighborhood or $C \cup \{x\}$ induces an irrelevant component, which are contradictions. Hence, D_C is non-empty. If $|A_C| \geq 2$, then we are done by Observation 18(ii). ◀

▶ **Lemma 21.** *Let $C \in \mathcal{C}_{\geq 3}$. Then, the number of vertices in C which are adjacent to at least one vertex in $B_C \cup D_C$ is at most $4k - 1$.*

Proof. By Observation 20, $|A_C| \geq 1$. Since $|V_X| \leq 4k$, $|D_C| \leq 4k-1$. Let C' be the set of vertices in C which are adjacent to $B_C \cup D_C$. For every vertex $v \in C'$, by Lemma 19, if B_v is non-empty, then D_v is non-empty. Since $v \in C'$, if B_v is empty, then also D_v is non-empty. For any two vertices $v, u \in C'$, by Observation 18(iii), $D_u \cap D_v = \emptyset$. Therefore $|C'| \leq |D_C| \leq 4k-1$. ◀

Now, we state the last rule of the kernelization.

▶ **Rule 5** (Clique Reduction). *Let $C \in \mathcal{C}_{\geq 3}$ such that $|C| > 4k$. Let C' be $C \cup A_C$. Let C'' be the set of vertices in C which are local to C'. Then, delete any $|C''|-1$ vertices from C''.*

▶ **Observation 22.** *After the application of clique reduction rule, the number of vertices retained in C is at most $4k$.*

Proof. By Lemma 21, the number of vertices in C which are not local to C' is at most $4k-1$. Hence, the rest of the vertices in C are local to C' in G. If $|C| > 4k$, clique reduction rule retains only one local vertex and delete all other vertices in C local to C'. ◀

▶ **Lemma 23.** *Clique reduction rule is safe and can be applied in polynomial time.*

Now we give the kernelization algorithm.

Kernelization of Diamond-free Edge Deletion

Let (G, k) be the input.

Step 1: Apply Phase 1 on (G, k) to obtain (G_1, k_1).

Step 2: Find X, V_X and $\mathcal{C}$ of G_1. Apply clique reduction rule on (G_1, k_1) to obtain (G', k_1).

Step 3: If neither Step 1 nor Step 2 is applicable on (G', k_1), then return (G', k_1). Otherwise apply the kernelization on (G', k_1).

▶ **Lemma 24.** *The kernelization algorithm is safe and can be applied in polynomial time.*

2.1 Bounding the Kernel Size

In this subsection, we bound the number of vertices in the kernel obtained by the kernelization. Let (G, k) be an instance of Diamond-free Edge Deletion and (G', k') is obtained by the kernelization. Consider an X, V_X and $\mathcal{C}$ of (G', k').

▶ **Lemma 25.** $\sum_{C \in \mathcal{C}_1} |C| = O(k^3)$.

Proof. Since, Phase 1 is not applicable on (G', k'), by Lemma 8(i), every vertex is a core member of G'. Let $\{v\} \in \mathcal{C}_1$. By Observation 20, v must be adjacent to two vertices $x, y \in V_X$ such that x and y are adjacent. Now consider the edge $\{x, y\}$. In the common neighborhood of $\{x, y\}$ there can be at most $2k+1$ vertices v with the property that $\{v\} \in \mathcal{C}_1$ (otherwise sunflower rule applies). Since there are at most $O(k^2)$ edges in $G'[V_X]$, we obtain that the total number of vertices in the singleton sets of $\mathcal{C}$ is $O(k^3)$. ◀

▶ **Lemma 26.**

(i) *Consider any two vertices $x, y \in V_X$. Let $\mathcal{C}' \subseteq \mathcal{C}_2 \cup \mathcal{C}_{\geq 3}$ such that for any $C \in \mathcal{C}'$, $x, y \in A_C$. If $\{x, y\} \in X$ then $|\mathcal{C}'| \leq 2k+1$. If $\{x, y\} \notin X$, then $|\mathcal{C}'| \leq 1$.*

(ii) *Consider any ordered pair of vertices (x, y) in V_X such that x and y are adjacent in G'. Let $\mathcal{C}' \subseteq \mathcal{C}_2 \cup \mathcal{C}_{\geq 3}$ such that for any $C \in \mathcal{C}'$, $x \in A_C$ and $y \in D_C$. If $\{x, y\} \in X$ then $|\mathcal{C}'| \leq 2k+1$. If $\{x, y\} \notin X$, then $|\mathcal{C}'| = 0$.*

Proof. (i). Let $C_a, C_b \in \mathcal{C}'$. By Observation 14(i), $|C_a \cap C_b| \leq 1$. If $v \in C_a \cap C_b$, then by Observation 14(ii), there is no edge between $C_a \setminus \{v\}$ and $C_b \setminus \{v\}$. Hence, $\{x, v, a, b\}$ induces a diamond where $a \in C_a \setminus \{v\}$ and $b \in C_b \setminus \{v\}$, which is edge disjoint with X, a contradiction. Hence $C_a \cap C_b = \emptyset$. Now, consider any two vertices $a \in C_a$ and $b \in C_b$. Clearly, $\{x, y, a, b\}$ induces a diamond. Hence, $\{x, y\}$ must be an edge in X, otherwise the diamond is edge disjoint with X, a contradiction. Therefore, if $\{x, y\} \notin X$, $|\mathcal{C}'| \leq 1$. Now we consider the case in which $\{x, y\} \in X$. If $|\mathcal{C}'| \geq 2k + 2$, we get at least $k + 1$ diamonds where every two diamonds have the only edge intersection $\{x, y\}$. Then sunflower rule applies, which is a contradiction.

(ii). Let $\mathcal{C}'$ be the set of all $C \in \mathcal{C}_2 \cup \mathcal{C}_{\geq 3}$ such that $x \in A_C$ and $y \in D_C$. Consider any two of them - C_a and C_b. By Observation 14(i), $|C_a \cap C_b| \leq 1$. If $v \in C_a \cap C_b$, then by Observation 14(ii), there is no edge between $C_a \setminus \{v\}$ and $C_b \setminus \{v\}$. Let $a \in C_a \setminus \{v\}$ and $b \in C_b \setminus \{v\}$. Then $\{x, v, a, b\}$ induces a diamond which is edge disjoint with X, a contradiction. Hence $C_a \cap C_b = \emptyset$. Let $a, a' \in C_a$ such that a is adjacent to y. Then, if $\{x, y\} \notin X$, $\{x, a, a', y\}$ induces a diamond, which is edge disjoint with X. Therefore, if $\{x, y\} \notin X$, then $|\mathcal{C}'| = 0$. Now we consider the case in which $\{x, y\} \in X$. If $|C'| \geq 2k + 2$, we get at least $k + 1$ diamonds where every two diamonds have the only edge intersection $\{x, y\}$. Then sunflower rule applies, which is a contradiction. ◀

▶ **Lemma 27.** $\sum_{C \in \mathcal{C}_2 \cup \mathcal{C}_{\geq 3}} |C| = O(k^3)$.

Proof. Consider any two adjacent vertices $x, y \in V_X$. Let $\mathcal{C}'_{xy} \subseteq \mathcal{C}_2 \cup \mathcal{C}_{\geq 3}$ be such that $x, y \in A_C$. Then by Lemma 26(i), if $\{x, y\} \in X$, then $|C'_{xy}| \leq 2k + 1$ and if $\{x, y\} \notin X$, then $|C'_{xy}| \leq 1$. Since there are at most $5k$ edges in X and $O(k^2)$ edges in $G[V_X] \setminus X$, $\bigcup_{\{x,y\} \in E(G[V_X])} C'_{xy}$ has at most $O(k) \cdot (2k + 1) + O(k^2) = O(k^2)$ maximal cliques. Since every maximal clique has at most $4k$ vertices (by Observation 22), the total number of vertices in those cliques is $O(k^3)$.

Now, let $\mathcal{C}'_{xy} \subseteq \mathcal{C}_2 \cup \mathcal{C}_{\geq 3}$ be such that $x \in A_C$ and $y \in D_C$. Then by Lemma 26(ii), if $\{x, y\} \in X$, then $|C'_{xy}| \leq 2k + 1$ and if $\{x, y\} \notin X$, then $|C'_{xy}| = 0$. Since there are at most $2 \cdot 5k = 10k$ ordered adjacent pairs of vertices in X, $\bigcup_{\{x,y\} \in E(G[V_X])} C'_{xy}$ has at most $O(k) \cdot (2k + 1)$ maximal cliques. Since every maximal clique has at most $4k$ vertices (by Observation 22), the total number of vertices in those cliques is $O(k^3)$.

Since, by Observation 20, for every $C \in \mathcal{C}$, there exist two vertices $x \in A_C$ and $y \in A_C \cup D_C$, we have counted every $C \in \mathcal{C}_2 \cup \mathcal{C}_{\geq 3}$. Hence $\sum_{C \in \mathcal{C}_2 \cup \mathcal{C}_{\geq 3}} |C| = O(k^3)$. ◀

▶ **Theorem 28.** *Given an instance (G, k) of* Diamond-free Edge Deletion*, the kernelization gives an instance (G', k) such that $|V(G')| = O(k^3)$ and $k' \leq k$ or declares that the instance is a no-instance.*

Proof. None of the rules increases the parameter k. Then, the theorem follows from Lemma 25 and Lemma 27 and the fact that $|V_X| \leq 4k$. ◀

3 Parameterized Lower Bound

Exponential Time Hypothesis (ETH) (along with Sparsification Lemma [14]) is an assumption that there is no algorithm which solves 3-SAT in time $2^{o(n+m)}(n + m)^{O(1)}$, where n is the number of variables and m is the number of clauses. We can use linear parameterized reduction from 3-SAT (with parameter $n + m$) to another parameterized problem to show that the latter does not have a subexponential parameterized algorithm, unless ETH fails.

In this section, we give a linear parameterized reduction from Vertex Cover on cubic (i.e., every vertex has degree 3) graphs to Diamond-free Edge Deletion. It has been proved that Vertex Cover is NP-complete on graphs with degree at most three [12] and on cubic planar graphs [16]. The reduction in [16] does not imply that there exists no parameterized subexponential time algorithm for Vertex Cover on cubic graphs. But, by modifying the reduction in [12] by using an insight from the reduction in [16], it can be easily proved that Vertex Cover is NP-complete on cubic graphs and cannot be solved in parameterized subexponential time, unless ETH fails.

▶ **Lemma 29.** Vertex Cover *is* NP-complete *on cubic graphs and cannot be solved in time* $2^{o(k)} \cdot |G|^{O(1)}$*, unless ETH fails.*

Proof Idea. A reduction from 3-SAT to Vertex Cover is given in [12] such that the input 3-SAT instance with n variables and m clauses is satisfiable if and only if the output graph has a vertex cover of size at most $5m$. The output graph has exactly $3m$ vertices with degree 2 and $6m$ vertices with degree 3. In order to make sure that every vertex has degree 3, we can use a technique used in [16] to convert a degree 2 vertex by a simple structure so that the 3-SAT instance is satisfiable if and only if the resultant graph has a vertex cover of size at most $11m$. ◀

Now we give a linear parameterized reduction from Vertex Cover on cubic graphs to Diamond-free Edge Deletion.

Reduction: Let (G, k) be an instance of Vertex Cover and let G be a cubic graph. We replace each edge uv of G by a path of length 3. For every edge uv, we denote the newly introduced vertices as s_{uv} and s_{vu} where s_{uv} is adjacent to u and s_{vu} is adjacent to v. Let S be the set of all new vertices. For every $u \in V(G)$, S_u denotes the three vertices in S adjacent to u. Make every pair of vertices in S_u adjacent to each other such that the vertices in S_u form a triangle. We introduce a universal vertex w which is adjacent to all the vertices in $V(G) \cup S$. For every edge uv in G, we make sure that the edge $s_{uv}s_{vu}$ is un-deletable by making it part of a large clique such that deleting $s_{uv}s_{vu}$ will create unmanageable number of diamonds. For this purpose we introduce a set $C_{\{u,v\}}$ of $6k$ vertices each of them are adjacent to each other and to both s_{uv} and s_{vu}. This completes the reduction. Let the resultant graph be G'.

For every vertex $u \in V(G)$, by G'_u we denote a subgraph of G' induced by $S_u \cup \{u, w\}$. G' when G is a K_4 is given in Figure 3. We will prove that (G, k) is a yes-instance if and only if $(G', 3k)$ is a yes-instance. Before proving this, we observe some properties of $(G', 3k)$.

▶ **Lemma 30.**

(i) *Let E' be a solution of size at most $3k$ of $(G', 3k)$. Then E' does not contain any edge from the graph induced by $C_{\{u,v\}} \cup \{s_{uv}, s_{vu}\}$.*

(ii) *Every induced diamond in G' contain the vertex w.*

Proof. (i). Let C' be $C_{\{u,v\}} \cup \{s_{uv}, s_{vu}\}$. Let $x, y \in C'$ be such that $e = \{x, y\} \in E'$. Consider any pair of vertices $x', y' \in C' \setminus \{x, y\}$. Clearly $\{x', y', x, y\}$ induces a diamond in $G' - e$. Any other pair of vertices $x'', y'' \in C' \setminus \{x, y\}$ such that $\{x', y'\} \cap \{x'', y''\} = \emptyset$ induces a diamond $\{x'', y'', x, y\}$ which is edge disjoint with that induced by $\{x', y', x, y\}$. There should be at least one edge in E' from every such diamond. Since there are $6k$ vertices in $C' \setminus \{x, y\}$, $|E'| \geq 3k + 1$, which is a contradiction.

(ii). Let H be $G' - w$. We claim that H is diamond-free. For every $u \in V(G)$, we observe that $S_u \cup \{u\}$ forms a maximal clique of H. For every pair of adjacent vertices $\{u, v\}$ in G,

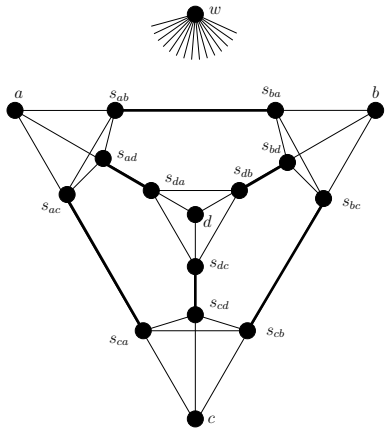


Figure 3 Graph G' when G is a K_4. w is adjacent to all visible vertices. A thick edge $s_{uv}s_{vu}$ denotes a clique of size $6k+2$ with the vertices $C_{\{u,v\}} \cup \{s_{uv}, s_{vu}\}$. s_{uv} and s_{vu} retain its adjacency as shown in the figure, whereas $C_{\{u,v\}}$ vertices are adjacent to only the vertices in $C_{\{u,v\}} \cup \{s_{uv}, s_{vu}\}$.

$C_{\{u,v\}} \cup \{s_{uv}, s_{vu}\}$ forms a maximal clique of H. Now, every edge in H is in one of these maximal cliques. Hence, by Lemma 11, H is diamond-free. ◀

▶ **Theorem 31.** Diamond-free Edge Deletion *is* NP-complete. *Furthermore, the problem cannot be solved in time* $2^{o(k)} \cdot |V(G)|^{O(1)}$*, unless ETH fails.*

Proof. Diamond-free Edge Deletion is trivially in NP. Let (G, k) be an instance of Vertex Cover on cubic graphs and we apply the reduction described to obtain $(G', 3k)$, an instance of Diamond-free Edge Deletion. We need to prove that (G, k) is a yes-instance of Vertex Cover if and only if $(G', 3k)$ is a yes-instance of Diamond-free Edge Deletion.

Let U be a vertex cover of size at most k of G. Let $D = \{s_{uv}w : u \in U, uv \in E(G)\}$, i.e., D is the set of edges between w and S_u for all $u \in U$. We claim that $G' - D$ is diamond-free. To prove this, we give a maximal clique partitioning of $G' - D$. For every vertex $u \in U$, $S_u \cup \{u\}$ is a maximal clique in $G' - D$. For every vertex $v \in V(G) \setminus U$, G'_v is a maximal clique in $G' - D$. For every edge $\{u, v\}$ in G, $C_{\{u,v\}} \cup \{s_{uv}, s_{vu}\}$ is a maximal clique in $G' - D$. Now, we observe that every edge in $G' - D$ is part of some maximal cliques obtained above. Since G is cubic, $|D| \leq 3k$.

Conversely, assume that D is the set of edges in G' such that $G' - D$ is diamond-free and $|D| \leq 3k$. For an edge $\{u, v\}$ in G, $\{s_{uv}, s_{vu}, w, c\}$, where c is any vertex in $C_{\{u,v\}}$ induces a diamond in G'. Since the only deletable edges in this diamond are $s_{uv}w$ and $s_{vu}w$, either of them, say $s_{uv}w$ must be in D. In that case, we observe that at least 2 more edges have to be deleted from G'_u. This implies that, if at all a single edge is deleted from G'_u, then at least 3 edges must be deleted from G'_u. Hence for every edge $uv \in E(G)$ at least 3 edges from G'_u or 3 edges from G'_v must be in D. Now let $U = \{u : D \text{ has an edge from } G'_u\}$. Clearly, U is a vertex cover of size at most k. ◀

4 Concluding Remarks

We obtained an $O(k^3)$ kernel for Diamond-free Edge Deletion which is an improvement over the previously known kernel. We gave an alternative proof for the NP-completeness

of Diamond-free Edge Deletion. We observed that the problem cannot be solved in parameterized subexponential time unless ETH fails. We believe that the vertex split rule introduced in this paper will be useful in similar settings. One way of extending our result is to give a polynomial kernel for $\mathcal{H}$-free Edge Deletion where $\mathcal{H}$ is a finite set of graphs containing diamond. We conclude with an open problem: Does Paw-free Edge Deletion admit a polynomial kernel? It is known that a graph is paw-free if and only if every component of it is either triangle-free or complete multipartite [17]. Since this characterization is easier compared to that of claw-free graphs, we believe that finding a polynomial kernel for Paw-free Edge Deletion will be easier compared to that of Claw-free Edge Deletion.

References

1. N. R. Aravind, R. B. Sandeep, and Naveen Sivadasan. On polynomial kernelization of $\mathcal{H}$-free edge deletion. In *Parameterized and Exact Computation*, pages 28–38. Springer International Publishing, 2014.
2. Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine Learning*, 56(1-3):89–113, 2004.
3. Hans L. Bodlaender and Babette van Antwerpen-de Fluiter. On intervalizing k-colored graphs for DNA physical mapping. *Discrete Applied Mathematics*, 71(1-3):55–77, 1996.
4. Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Inf. Process. Lett.*, 58(4):171–176, 1996.
5. Leizhen Cai and Yufei Cai. Incompressibility of H-free edge modification problems. *Algorithmica*, 71(3):731–757, 2015.
6. Yufei Cai. Polynomial kernelisation of H-free edge modification problems. Mphil thesis, Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong SAR, China, 2012.
7. Marek Cygan, Marcin Pilipczuk, Michał Pilipczuk, Erik Jan van Leeuwen, and Marcin Wrochna. Polynomial kernelization for removing induced claws and diamonds. In *WG*, 2015.
8. Pål Grønås Drange and Michał Pilipczuk. A polynomial kernel for trivially perfect editing. In *ESA*, 2015.
9. Ehab S El-Mallah and Charles J Colbourn. The complexity of some edge deletion problems. *Circuits and Systems, IEEE Transactions on*, 35(3):354–362, 1988.
10. A Farrugia. Clique-helly graphs and hereditary clique-helly graphs, a mini-survey. *Algoritmic graph theory (CS 762)–2002–Project, Dept. of Comb., University of Waterloo*, 2002.
11. Michael R. Fellows, Jiong Guo, Christian Komusiewicz, Rolf Niedermeier, and Johannes Uhlmann. Graph-based data clustering with overlaps. *Discrete Optimization*, 8(1):2–17, 2011.
12. M. R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some simplified NP-complete graph problems. *Theor. Comput. Sci.*, 1(3):237–267, 1976.
13. Paul W. Goldberg, Martin Charles Golumbic, Haim Kaplan, and Ron Shamir. Four strikes against physical mapping of DNA. *Journal of Computational Biology*, 2(1):139–152, 1995.
14. Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.
15. Stefan Kratsch and Magnus Wahlström. Two edge modification problems without polynomial kernels. *Discrete Optimization*, 10(3):193–199, 2013.
16. Bojan Mohar. Face covers and the genus problem for apex graphs. *J. Comb. Theory, Ser. B*, 82(1):102–117, 2001.
17. Stephan Olariu. Paw-fee graphs. *Inf. Process. Lett.*, 28(1):53–54, 1988.

On Kernelization and Approximation for the Vector Connectivity Problem*

Stefan Kratsch[1] and Manuel Sorge[2]

1 **University of Bonn, Germany**
kratsch@cs.uni-bonn.de
2 **Technical University Berlin, Germany**
manuel.sorge@tu-berlin.de

Abstract

In the VECTOR CONNECTIVITY problem we are given an undirected graph $G = (V, E)$, a demand function $\phi\colon V \to \{0, \dots, d\}$, and an integer k. The question is whether there exists a set S of at most k vertices such that every vertex $v \in V \setminus S$ has at least $\phi(v)$ vertex-disjoint paths to S; this abstractly captures questions about placing servers in a network, or warehouses on a map, relative to demands. The problem is NP-hard already for instances with $d = 4$ (Cicalese et al., Theor. Comput. Sci. '15), admits a log-factor approximation (Boros et al., Networks '14), and is fixed-parameter tractable in terms of k (Lokshtanov, unpublished '14).

We prove several results regarding kernelization and approximation for VECTOR CONNECTIVITY and the variant VECTOR d-CONNECTIVITY where the upper bound d on demands is a constant. For VECTOR d-CONNECTIVITY we give a factor d-approximation algorithm and construct a vertex-linear kernelization, i.e., an efficient reduction to an equivalent instance with $f(d)k = \mathcal{O}(k)$ vertices. For VECTOR CONNECTIVITY we get a factor opt-approximation and we show that it has no kernelization to size polynomial in $k+d$ unless NP $\subseteq$ coNP/poly, making $f(d)\,\mathrm{poly}(k)$ optimal for VECTOR d-CONNECTIVITY. Finally, we provide a write-up for fixed-parameter tractability of VECTOR CONNECTIVITY(k) by giving a different algorithm based on matroid intersection.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases parameterized complexity, kernelization, approximation

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.377

1 Introduction

In the VECTOR CONNECTIVITY problem we are given an undirected graph $G = (V, E)$, a demand function $\phi\colon V \to \{0, \dots, d\}$, and an integer $k \in \mathbb{N}$. The question is whether there exists a set S of at most k vertices of G such that every vertex $v \in V$ is either in S or has at least $\phi(v)$ vertex-disjoint paths to vertices in S; the paths may share the vertex v itself.

VECTOR CONNECTIVITY
Input: A graph $G = (V, E)$, a function $\phi\colon V \to \{0, \dots, d\}$, $k \in \mathbb{N}$.
Question: Is there a set S of at most k vertices such that each vertex $v \in V \setminus S$ has $\phi(v)$ vertex-disjoint paths with endpoints in S?

The value $\phi(v)$ is also called the *demand* of vertex v. We call $S \subseteq V$ a *vector connectivity set* for (G, ϕ), if it fulfills the requirements above. We do not formally distinguish decision and optimization version; k is not needed for the latter.

* Supported by the German Research Foundation (DFG) under grants KR 4286/1-1 and NI 369/12-2.

For intuition about the problem formulation and applications one may imagine a logistical problem about placing warehouses to service locations on a map (or servers in a network): Each location has a particular demand, which could capture volume and redundancy requirements or level of importance. Demand can be satisfied by placing a warehouse at the location or by ensuring that there are enough disjoint paths from the location to warehouses; both can be seen as guaranteeing sufficient throughput or ensuring access that is failsafe up to demand minus one interruptions in connections. In this way, VECTOR CONNECTIVITY can also be seen as a variant of the FACILITY LOCATION problem in which the costs of serving demand are negligible, but instead redundancy is required.

Related work. The study of the VECTOR CONNECTIVITY problem was initiated recently by Boros et al. [2] who gave polynomial-time algorithms for trees, cographs, and split graphs. Moreover, they obtained an $\ln n + 2$-factor approximation algorithm for the general case. More recently, Cicalese et al. [3] continued the study of VECTOR CONNECTIVITY and amongst other results proved that it is APX-hard (and NP-hard) on general graphs, even when all demands are upper bounded by four. In a related talk during a recent Dagstuhl seminar (Dagstuhl Seminar 14071 on "Graph Modification Problems.") Milanič asked whether VECTOR CONNECTIVITY is fixed-parameter tractable with respect to the maximum solution size k. This was answered affirmatively by Lokshtanov (unpublished).

Our results. We obtain results regarding kernelization and approximation for VECTOR CONNECTIVITY and VECTOR d-CONNECTIVITY where the maximum demand d is a fixed constant. We also provide a self-contained write-up for fixed-parameter tractability of VECTOR CONNECTIVITY with parameter k (Sec. 4); it is different from Lokshtanov's approach and instead relies on a matroid intersection algorithm of Marx [8].

Our analysis of the problem starts with a new data reduction rule stating that we can safely "forget" the demand of $r := \phi(v)$ at a vertex v if v has vertex-disjoint paths to r vertices each of demand at least r (Sec. 2). After exhaustive application of the rule, all remaining vertices of demand r must have cuts of size at most $r - 1$ separating them from other vertices of demand at least r. By analyzing these cuts we then show that any yes-instance of VECTOR d-CONNECTIVITY can have at most d^2k vertices with nonzero demand; the corresponding bound for VECTOR CONNECTIVITY is $k^3 + k$. (Both bounds also hold when replacing k by the optimum cost opt). This directly would yield factor d^2 and factor $\mathsf{opt}^2 + 1$ approximation algorithms. We improve upon this in Sec. 3 by giving a variant of the reduction rule that works correctly relative to any partial solution S_0, which can then be applied in each round of our approximation algorithm. The algorithm follows the local-ratio paradigm and, surprisingly perhaps, proceeds by always selecting a vertex of *lowest* demand. We thus obtain ratios of d and opt respectively, i.e., the returned solution is of size at most $d \cdot \mathsf{opt}$ for VECTOR d-CONNECTIVITY and at most opt^2 for VECTOR CONNECTIVITY.

Regarding kernelization we show in Sec. 6 that there is no kernel with size polynomial in k or even $k + d$ for VECTOR CONNECTIVITY unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$ (and the polynomial hierarchy collapses); our proof also implies that the problem is $\mathsf{WK}[1]$-hard (cf. [6]). Nevertheless, when d is a fixed constant, we prove that a vertex-linear kernelization is possible. A non-constructive proof of this can be pieced together from recent work on meta kernelization on sparse graph classes (see below). Instead we give a constructive algorithm building on an explicit (though technical) description of what constitutes equivalent subproblems. We also have a direct proof for the number of such subproblems in an instance with parameter k rather than relying on a known argument for bounding the number of connected subgraphs

of bounded size and bounded neighborhood size (via the two-families theorem of Bollobas, cf. [7]). A brief overview of our kernelization is given in Sec. 5. The bound of $f(d)k = \mathcal{O}(k)$ vertices is optimal in the sense that the lower bound for parameter $d + k$ rules out total size $\text{poly}(k + d)$, i.e. we need to allow superpolynomial dependence on d.

A non-constructive kernelization argument. The mentioned results on meta kernelization for problems on sparse graph classes mostly rely on the notion of a protrusion, i.e., an induced subgraph of bounded treewidth such that only a bounded number of its vertices have neighbors in the rest of the graph (the *boundary*). Under appropriate assumptions there are general results that allow the replacements of protrusions by equivalent ones of bounded size, which yields kernelization results assuming that the graph can be decomposed into a bounded number of protrusions. Intuitively, having small boundary size limits the interaction of a protrusion with the remaining graph. The assumption of bounded treewidth ensures fast algorithms for solving subproblems on the protrusion and also leads to fairly general arguments for obtaining small equivalent replacements. Note that for VECTOR CONNECTIVITY there is no reason to assume small treewidth and we also do not restrict the input graphs.

Arguably, the most crucial properties of a protrusion are the small boundary and the fact that we can efficiently compute subproblems on the protrusion; in principle, we do not need bounded treewidth. (Intuitively, we want to know the best solution value for each choice of interaction of a global solution with the boundary of the protrusion.) Thus, it seems natural to define a relaxed variant of protrusions by insisting on a small boundary and efficient algorithms for solving subproblems rather than demanding bounded treewidth. Fomin et al. [5] follow this approach for problems related to picking a certain subset of vertices like DOMINATING SET: Their relaxed definition of a r-DS-protrusion requires a boundary of size at most r and the existence of a solution of size at most r for the protrusion itself; the latter part implies efficient algorithms since we can afford to simply try all $r = \mathcal{O}(1)$ sized vertex subsets. Fomin et al. also derive a general protrusion replacement routine for problems that have finite integer index (a common assumption for meta kernelization, see, e.g., Fomin et al. [5]) and are monotone when provided also with a sufficiently fast algorithm, like the one implied by having a solution of size at most r for the protrusion. Fomin et al. [5] remark that the procedure is not constructive since it assumes hard-wiring an appropriate set of representative graphs, whose existence is implied by being finite integer index.

From previous work of Cicalese [3] it is known that VECTOR CONNECTIVITY is an implicit hitting set problem where the set family consists of all connected subgraphs with neighborhood size smaller than the largest demand in the set; the family is exponential size, but we get size roughly $\mathcal{O}(n^d)$ when demands are at most d. The procedure of Fomin et al. [5] can be applied to minimal sets in this family and will shrink them to some size bounded by an unknown function in d, say $h(d)$. Then one can apply the two-families theorem of Bollobas (cf. [7]) to prove that each vertex is contained in at most $\binom{h(d)+d}{d}$ such sets. Because the solution must hit all sets with k vertices, a yes-instance can have at most $k \cdot \binom{h(d)+d}{d}$ sets. This argument can be completed to a vertex-linear kernelization.

In comparison, we obtain an explicit upper bound of $d^2k \cdot 2^{d^3+d}$ for the number of subproblems that need to be replaced, by considering a set family that is different from the implicit hitting set instance (but contains supersets of all those sets). We also have a constructive description of what constitutes equivalent subproblems. This enables us to give a single algorithm that works for all values of d based on maximum flow computations (rather than requiring for each value of d an algorithm with hard-wired representative subproblems).

Preliminaries. We use standard graph notation. Apart from this, due to the nature of the VECTOR CONNECTIVITY problem, we are frequently interested in disjoint paths from a vertex v to some vertex set S, where the paths are vertex-disjoint except for sharing v. The natural counterpart, in the spirit of Menger's theorem, are *v, S-separators* $C \subseteq V(G) \setminus \{v\}$ such that in $G - C$ no vertex of $S \setminus C$ is reachable from v; the maximum number of disjoint paths from v to S that may overlap in v equals the minimum size of a v, S-separator. Throughout, by disjoint paths from v to S, or v, S-separator (for any single vertex v and any vertex set S) we mean the mentioned path packings and separators with special role of v. Many proofs use the function $f\colon 2^V \to \mathbb{N}\colon U \mapsto |N(U)|$, which is well-known to be *submodular*, i.e., for all $X, Y \subseteq V$ we have $f(X) + f(Y) \geq f(X \cap Y) + f(X \cup Y)$.

So-called *closest sets* will be used frequently; these occur naturally in cut problems but appear to have no generalized name. We define a vertex set C to be *closest to* v if C is the unique v, C-separator of size at most $|C|$, where v, C-separator is in the above sense. As an example, if C is a minimum s, t-vertex cut that, amongst such cuts, has the smallest connected component for s in $G - C$ then C is also closest.

2 Reducing the number of demand vertices

In this section we introduce a reduction rule for VECTOR CONNECTIVITY that reduces the total demand. We prove that the reduction rule does not affect the solution space, which makes it applicable not only for kernelization but also for approximation and other techniques. In Section 5, we will use this rule in our kernelization for VECTOR d-CONNECTIVITY(k). In the following two sections, as applications of these reduction rules and the insights gained we get approximation algorithms for VECTOR d-CONNECTIVITY and VECTOR CONNECTIVITY, and an alternative FPT algorithm for VECTOR CONNECTIVITY(k).

▶ **Rule 1.** *Let (G, ϕ, k) be an instance of* VECTOR CONNECTIVITY. *If a vertex $v \in V$ has at least $\phi(v)$ vertex-disjoint paths to vertices different from v with demand at least $\phi(v)$ then set the demand of v to zero.*

We prove that the rule does not affect the space of feasible solutions.

▶ **Lemma 2.** *Let (G, ϕ, k) be an instance of* VECTOR CONNECTIVITY *and let (G, ϕ', k) be the instance obtained via a single application of Rule 1. Every $S \subseteq V(G)$ is a solution for (G, ϕ, k) if and only if it is a solution for (G, ϕ', k).*

Proof. Let v denote the vertex whose demand was set to zero by the reduction rule and define $r := \phi(v)$. Clearly, $\phi(u) = \phi'(u)$ for all vertices $u \in V(G) \setminus \{v\}$, and $\phi'(v) = 0$. It suffices to show that if S fulfills demands according to ϕ' then S fulfills also demands according to ϕ since $\phi(u) \geq \phi'(u)$ for all $u \in V(G)$. This in turn comes down to proving that S fulfills the demand of r at v assuming that it fulfills demands according to ϕ'. If $v \in S$ then the demand at v is trivially fulfilled so henceforth assume that $v \notin S$.

Let $w_1, \ldots, w_r$ denote vertices different from v with demand each at least r such that there exist r vertex-disjoint paths from v to $\{w_1, \ldots, w_r\}$, i.e., a single path to each w_i. Existence of such vertices is required for the application of the rule.

Assume for contradiction that S does not satisfy the demand of r at v (recall that $v \notin S$, by assumption), i.e., that there are no r vertex-disjoint paths from v to S that overlap only in v. It follows directly that there is a v, S-separator C of size at most $r - 1$. (Recall that C may contain vertices of S but not the vertex v.) Let R denote the connected component of v in $G - C$, then the following holds for each vertex $w_i \in \{w_1, \ldots, w_r\}$:

1. If $w_i \in S$ then $w_i \notin R$: Otherwise, we would have $S \cap R \supseteq \{w_i\} \neq \emptyset$ contradicting the fact that v can reach no vertex of S in $G - C$.
2. If $w_i \notin S$ then $w_i \notin R$: Since S fulfills demands according to ϕ' there must be at least r vertex-disjoint paths from w_i to S that overlap only in w_i. However, since $w_i \in R$ the set C is also a w_i, S-separator; a contradiction since C has size less than r.

Thus, no vertex from $w_1, \ldots, w_r$ is contained in R. This, however, implies that C separates v from $\{w_1, \ldots, w_r\}$, contradicting the fact that there are r vertex-disjoint paths from v to $\{w_1, \ldots, w_r\}$ that overlap only in v. It follows that no such v, S-separator C can exist, and, hence, that there are at least $r = \phi(v)$ vertex-disjoint paths from v to S, as claimed. Thus, S fulfills the demand of r at v and hence all demand according to ϕ. (Recall that the converse is trivial since $\phi(u) \geq \phi'(u)$ for all vertices $u \in V(G)$.) ◀

The idea for applying Rule 1 in polynomial time is to use a maximum-flow computation with v as source and all other vertices with nonzero demand as sinks.

To analyze the impact of Rule 1 we will now bound the number of nonzero demand vertices in an exhaustively reduced instance in terms of the optimum solution size opt and the maximum demand d. To this end, we require the technical lemma below about the structure of reduced instances as well as some notation. If (G, ϕ, k) is reduced according to Rule 1 then for each vertex v with demand $r = \phi(v) \geq 1$ there is a cut set C of size at most $r - 1$ that separates v from all other vertices with demand at least r. We fix for each vertex v with demand at least one a vertex set C, denoted $C(v)$, by picking the unique closest minimum v, D_v-separator where $D_v = \{u \in V \setminus \{v\} \mid \phi(u) \geq \phi(v)\}$. Furthermore, for such vertices v, let $R(v)$ denote the connected component of v in $G - C(v)$.

Clearly, a solution S must intersect each $R(v)$ since $|C(v)| < \phi(v)$. The following lemma shows implicitly that Rule 1 limits the amount of overlap of sets $R(v)$. Intuitively, sets $R(v)$ must overlap in order to "share" solution vertices.

▶ **Lemma 3.** *Let (G, ϕ, k) be reduced under Rule 1. Let $u, v \in V(G)$ be distinct vertices with $\phi(u) = \phi(v) \geq 1$. If $R(u) \cap R(v) \neq \emptyset$ then $u \in C(v)$ or $v \in C(u)$.*

Now, we give the promised bound on the number of nonzero demand vertices.

▶ **Lemma 4.** *Let (G, ϕ, k) be an instance of* VECTOR CONNECTIVITY *that is reduced under Rule 1 and let opt denote the minimum size of feasible solutions $S \subseteq V$. Then there are at most $d^2\mathsf{opt}$ nonzero demand vertices in G.*

Lemma 4 directly implies reduction rules for VECTOR d-CONNECTIVITY(k) and VECTOR CONNECTIVITY(k): For the former, if there are more than d^2k vertices then opt must exceed k and we can safely reject the instance. For the latter, there can be at most k vertices of demand greater than k since those must be in the solution. Additionally, if $\mathsf{opt} \leq k$ then there are at most $d^2\mathsf{opt} \leq k^3$ vertices of demand at most $d = k$; a total of $k^3 + k$.

We spell out the rule for VECTOR d-CONNECTIVITY(k) because it is used in our kernelization. The bound of $k^3 + k$ for VECTOR CONNECTIVITY(k) is crucial for our FPT-algorithm.

▶ **Rule 5.** *Let (G, ϕ, k) be reduced under Rule 1, with $\phi \colon V(G) \to \{0, \ldots, d\}$. If there are more than d^2k vertices of nonzero demand return* NO.

3 Approximation algorithm

In this section we discuss the approximability of VECTOR d-CONNECTIVITY. We know from Lemma 4 that the number of vertices with nonzero demand is at most $d^2\mathsf{opt}$ where opt

denotes the minimum size solution for the instance in question. This directly implies a factor d^2-approximation because taking all nonzero demand vertices constitutes a feasible solution. We now show how to improve this to a factor d-approximation for VECTOR d-CONNECTIVITY.

The approximation algorithm will work as follows: We maintain an initially empty partial solution $S_0 \subseteq V$. In each round, we add at most d vertices to S_0 and show that this always brings us at least one step closer to a solution, i.e., the optimal number of required additional vertices shrinks by at least one. To achieve this, we update Rule 1 to take the partial solution S_0 into account.

▶ **Rule 6.** *Let (G, ϕ, k) be an instance of* VECTOR CONNECTIVITY *and let $S_0 \subseteq V(G)$. If there is a vertex v with non-zero demand and a vertex set W not containing v such that each vertex in W has demand at least $\phi(v)$ and v has at least $\phi(v)$ vertex-disjoint paths to $S_0 \cup W$, then set the demand of v to zero. Similarly, if $v \in S_0$ then also set its demand to zero.*

Intuitively, vertices in S_0 get the same status as vertices with demand at least $\phi(v)$ for applying the reduction argument. The proof of correctness now has to take into account that we seek a solution that includes S_0.

▶ **Lemma 7.** *Let (G, ϕ, k) be an instance of* VECTOR CONNECTIVITY*, let $S_0 \subseteq V(G)$, and let (G, ϕ', k) be the instance obtained via a single application of Rule 1. For every $S \subseteq V(G)$ it holds that $S \cup S_0$ is a solution for (G, ϕ, k) if and only if $S \cup S_0$ is a solution for (G, ϕ', k).*

It follows, that we can safely apply Rule 6, as a variant of Rule 1, in the presence of a partial solution S_0. It is easy to see that Rule 6 can be applied exhaustively in polynomial time because testing for any vertex v is a single two-way min-cut computation and each successful application lowers the number of nonzero demand vertices by one.

We now describe our approximation algorithm. The algorithm maintains an instance (G, ϕ), a set $S_0 \subseteq V(G)$, and an integer $\ell \in \mathbb{N}$. Given an instance (G, ϕ) the algorithm proceeds in rounds to build S_0, which will eventually be a complete (approximate) solution. We start with $S_0 = \emptyset$ and $\ell = 0$. In any single round, for given (G, ϕ), set S_0, and integer ℓ the algorithm proceeds as follows:

1. Exhaustively apply Rule 6 to (G, ϕ) and S_0, possibly changing ϕ.
2. If S_0 satisfies all demands of (G, ϕ) then return S_0 as a solution (and stop).
3. Otherwise, pick a vertex $v \in V(G)$ *of minimum nonzero demand*. Because we have exhaustively applied Rule 6 there must be a set C of less than $\phi(v) \leq d$ vertices that separates v from S_0 and all vertices of demand at least $\phi(v)$. Add $\{v\} \cup C$ to S_0 and increase ℓ by one. Note that we add at most $\phi(v) \leq d$ vertices to S_0 because $|C| < \phi(v)$.

After Step 3 the algorithm continues with Step 1. The following Invariant 8 guarantees that the algorithm runs for at most opt rounds. The approximation factor follows since only this step adds small "non-optimal" parts to the solution.

▶ **Invariant 8.** *There exists a set S_1 of at most $\mathsf{opt} - \ell$ vertices such that $S_0 \cup S_1$ is a feasible solution for (G, ϕ).*

▶ **Theorem 9.** *The* VECTOR d-CONNECTIVITY *problem admits a polynomial-time factor d-approximation.*

We can also derive an approximation algorithm for VECTOR CONNECTIVITY, where there is no fixed upper bound on the maximum demand. To this end, we can rerun the previous algorithm for all "guesses" of $\mathsf{opt}_0 \in \{1, \ldots, n\}$. In each run, we start with S_0 containing all

vertices of demand greater than the guessed value opt_0, since those must be contained in every solution of total size at most opt_0. Then the maximum demand is $d = \mathsf{opt}_0$ and we get a d-approximate set of vertices to add to S_0 to get a feasible solution. When $\mathsf{opt}_0 = \mathsf{opt}$, then opt must also include the same set S_0 and for the remaining $\mathsf{opt} - |S_0| \leq \mathsf{opt}$ vertices we have a d-approximate extension; we get a solution of total size at most opt^2.

▶ **Corollary 10.** *The* VECTOR CONNECTIVITY *problem admits a polynomial-time approximation algorithm that returns a solution of size at most* opt^2*, where* opt *denotes the optimum solution size for the input.*

4 FPT algorithm for Vector Connectivity(k)

In this section we present a randomized FPT-algorithm for VECTOR CONNECTIVITY(k). (We recall that Lokshtanov announced this to be FPT.) Recall that the reduction rules in Section 2 also allow us to reduce the number of nonzero demand vertices to at most $k^3 + k$ (or safely reject). Based on this we are able to give a randomized algorithm building on a randomized FPT algorithm of Marx [8] for intersection of linear matroids. (The randomization comes from the need to construct a representation for the required matroids.) Concretely, this permits us to search for an independent set of size k in $k^3 + k$ linear matroids over the same ground set, where the independent set corresponds to the desired solution and each single matroid ensures that one demand vertex is satisfied.

▶ **Theorem 11.** VECTOR CONNECTIVITY(k) *is randomized fixed-parameter tractable without false positives and error probability exponentially small in the input size.*

Proof Sketch. W.l.o.g., input instances (G, ϕ, k) have at most k^3+k nonzero demand vertices (else apply the reduction rules); let $D = \{v \in V(G) \mid \phi(v) \geq 1\}$. Clearly, if the instance is YES then there exist also solutions of size *exactly* k (barring $|V(G)| < k$ which is trivial).

Algorithm. As a first step, we guess the intersection of a solution S^* of size k with the set D; there are at most $(k^3 + k)^k$ choices for $S_0 = D \cap S^*$. All vertices of demand exceeding k must be contained in S_0 for S^* to be a solution.

For each $v \in D \setminus S_0$, we construct a matroid M_v over $V' = V \setminus D$ as follows.

1. Build a graph G_v by first adding to G additional $c-1$ copies of v, called $v_2, \ldots, v_c$, where $c = \phi(v)$, and use $v_1 := v$ for convenience. Second, add $r = k - c$ universal vertices $w_1, \ldots, w_r$ (i.e., neighborhood $V \cup \{v_2, \ldots, v_c\}$).
2. Let M'_v denote the gammoid on G_v with source set $T = \{v_1, \ldots, v_c, w_1, \ldots, w_r\}$ and ground set $V' \cup S_0$. Recall that the independent sets of a gammoid are exactly those subsets I of the ground set that have $|I|$ vertex-disjoint paths from the sources to I. Gammoids are linear matroids and a representation over a sufficiently large field can be found in randomized polynomial time (cf. [8]).
3. Create M_v from M'_v by contracting S_0, making its ground set V'. If S_0 is independent in M'_v then any I is independent in M_v if and only if $S_0 \cup I$ is independent in M'_v.

Use Marx' algorithm [8] to search for a set I^* of size $k - |S_0|$ that is independent in each matroid M_v for $v \in D \setminus S_0$. If a set I^* is found then test whether $S_0 \cup I^*$ is a vector connectivity set for (G, ϕ, k) by appropriate polynomial-time flow computations. If yes then return the solution $S_0 \cup I^*$. Otherwise, if $S_0 \cup I^*$ is not a solution or if no set I^* was found then try the next set S_0, or answer NO if no set S_0 is left to check.

Correctness. Clearly, if (G, ϕ, k) is NO then the algorithm will always answer NO as all possible solutions $S_0 \cup I^*$ are tested for feasibility.

Assume now that (G, ϕ, k) is YES, let S^* a solution of size k, and let $S_0 = D \cap S^*$. Note that $S^* \subseteq V' \cup S_0$. Pick any $v \in D \setminus S_0$. It follows that there are $c = \phi(v)$ paths from v to S^* in G that are vertex-disjoint except for v. Thus, in G_v we get c (fully) vertex-disjoint paths from $\{v_1, \ldots, v_c\}$ to S^*, by giving each path a private copy of v. We get additional $r = k - c$ paths from $\{w_1, \ldots, w_r\}$ to the remaining vertices of S^* since $S^* \subseteq V' \cup S_0 \subseteq N(w_i)$. Thus, the set S^* is independent in each gammoid M'_v. Therefore, in each M'_v also $S_0 \subseteq S^*$ is independent. This implies that in M_v (obtained by contraction of S_0) the set $S^* \setminus S_0$ is independent and has size $k - |S_0|$. Moreover, any I is independent in M_v if and only if $I \cup S_0$ is independent in M'_v. It follows, from the former statement, that Marx' algorithm will find *some* set I of size $k - |S_0|$ that is independent in all matroids M_v for $v \in D \setminus S_0$.

We claim that $I \cup S_0$ is a vector connectivity set for (G, ϕ, k). Let $v \in D \setminus S_0$. We know that I is independent in M_v and, thus, $S := I \cup S_0$ is independent in M'_v. Thus, in G_v there are $|S| = k$ paths from T to S. This entails $c = \phi(v)$ vertex-disjoint paths from $\{v_1, \ldots, v_c\}$ to S that each contain no further vertex of T since $|T| = k$. By construction of G_v, we directly get $\phi(v)$ paths from v to S in G that are vertex-disjoint except for overlap in v. Thus, S satisfies the demand of any $v \in D \setminus S_0$. Since $S \supseteq S_0$, we see that S satisfies all demands. Thus, the algorithm returns a feasible solution, as required.

Runtime. Marx' algorithm for finding a set of size k' that is independent in ℓ matroids has FPT running time with respect to $k' + \ell$. We have $k' \leq k$ and $\ell \leq |D| \leq k^3 + k$ in all iterations of the algorithm and there are at most $(k^3 + k)^k$ iterations. This gives a total time that is FPT with respect to k, as claimed. ◀

5 Vertex-linear kernelization for constant demand

In this section we give an outline of our vertex-linear kernelization for VECTOR d-CONNECTIVITY(k), i.e., with d a constant and k the parameter. We will focus on explaining how we can directly bound the number of subproblems without using Bollobas' two-families theorem, whose bound depends on the size of reduced subproblems. Then we give some intuition about our explicit definition for equivalent subproblems and how to replace them.

The starting point for our kernelization are Reduction Rules 1 and 5, and a result of Cicalese et al. [3] that relates vector connectivity sets for (G, ϕ) to hitting sets for a family of connected subgraphs of G: The family contains all sets X such that $G[X]$ is connected and some vertex $v \in X$ has demand larger than $|N(X)|$; intuitively, every solution S must intersect each set X. We use instead a family $\mathcal{X}(G, \phi)$ containing only the *minimal* sets X. For ease of presentation we define $D(G, \phi) := \{v \in V(G) \mid \phi(v) \geq 1\}$, and use the shorthand $D = D(G, \phi)$ whenever G and ϕ are clear from context.

▶ **Proposition 12** (adapted from Cicalese et al. [3, Prop. 1]). *Let $G = (V, E)$, let $\phi\colon V \to \mathbb{N}$, and let $\mathcal{X} := \mathcal{X}(G, \phi)$. Then every set $S \subseteq V$ is a vector connectivity set for (G, ϕ) if and only if it is a hitting set for $\mathcal{X}$, i.e., it has a nonempty intersection with each $X \in \mathcal{X}$.*

For the general case of VECTOR CONNECTIVITY with unrestricted demands the size of $\mathcal{X}(G, \phi)$ can be exponential in $|V(G)|$; for VECTOR d-CONNECTIVITY there is a straightforward bound of $|\mathcal{X}| = \mathcal{O}(|V(G)|^d)$ since $|N(X)| \leq d - 1$. However, even for VECTOR d-CONNECTIVITY, the sets $X \in \mathcal{X}$ are not necessarily small and, thus, we will not take a hitting set approach for the kernelization; we will not materialize the set $\mathcal{X}$ but use it only for analysis.

To arrive at our kernelization we will later establish a reduction rule that shrinks connected subgraphs with small boundary and bounded number of demand vertices to constant size. This is akin to black-box protrusion-based reduction rules, especially as in [5], but we give

an explicit algorithm that comes down to elementary two-way flow computations. To get an explicit (linear) bound for the number of subproblems, we introduce a new family $\mathcal{Y}$ with larger but (as we will see) fewer sets, and apply the reduction process to graphs $G[Y]$ with $Y \in \mathcal{Y}$. Alternatively, as pointed out in the introduction, one may use a result of Bollobas for bounding the number of sets in $\mathcal{X}$ once they are small; a caveat is that this bound would depend on the final size of sets in $\mathcal{X}$, whereas we have a direct and explicit bound for $|\mathcal{Y}|$.

▶ **Definition 13** ($\mathcal{Y}(G, \phi, d)$)**.** Let $G = (V, E)$, let $d \in \mathbb{N}$, and let $\phi\colon V \to \{0, \ldots, d\}$. The family $\mathcal{Y}(G, \phi, d)$ contains all sets $Y \subseteq V$ with

1. $G[Y]$ is connected,
2. $|Y \cap D| \leq d^3$, i.e., Y contains at most d^3 vertices v with nonzero demand,
3. $|N(Y)| \leq d$, i.e., Y has at most d neighbors, and
4. there is a vertex $v \in Y \cap D$, i.e., $\phi(v) \geq 1$, such that $N(Y)$ is the unique closest minimum $v, D \setminus Y$-separator.

We show that each set $X \in \mathcal{X}(G, \phi)$ is a subset of some $Y \in \mathcal{Y}(G, \phi, d)$.

▶ **Lemma 14.** *Let $G = (V, E)$, $d \in \mathbb{N}$, and $\phi\colon V \to \{0, \ldots, d\}$. Let $\mathcal{X} := \mathcal{X}(G, \phi)$ and $\mathcal{Y} := \mathcal{Y}(G, \phi, d)$. Then for all $X \in \mathcal{X}$ there exists $Y \in \mathcal{Y}$ with $X \subseteq Y$.*

We prove that the number of sets $Y \in \mathcal{Y}$ is linear in k for every fixed d, by analyzing a branching algorithm for finding $Y \in \mathcal{Y}$. Thus, by later shrinking the size of sets in $\mathcal{Y}$ to some constant we get $\mathcal{O}(k)$ vertices in total over sets $Y \in \mathcal{Y}$.

▶ **Lemma 15.** *Let (G, ϕ, k) an instance of* VECTOR d-CONNECTIVITY(k) *and let $\mathcal{Y} := \mathcal{Y}(G, \phi, d)$. Then $|\mathcal{Y}| \leq d^2 k \cdot 2^{d^3+d}$.*

5.1 Reducing the size of sets in $\mathcal{Y}$

Let us describe how to reduce the size of sets $Y \in \mathcal{Y}$ through modifications on the graph G. At a high level, this will be achieved by replacing subgraphs $G[Y]$ by "equivalent" subgraphs of bounded size. When this is done, we know that the total number of vertices in sets $Y \in \mathcal{Y}$ is $\mathcal{O}(k)$. Since this part is somewhat technical, the proof details are deferred to a full version.

Consider a set $Y \in \mathcal{Y}$ and its (small) neighborhood $Z := N_G(Y)$. Think of deciding whether (G, ϕ, k) is YES as a game between two players, Alice and Bob. Alice sees only $G[Y \cup Z]$ and wants to satisfy the demands of all vertices in Y, and Bob sees only $G - Y$ and wants to satisfy the demands of the vertices in $V \setminus Y$. To achieve a small solution the players must cooperate and exchange information about paths between vertices in Z, or between Z and vertices of a partial solution, that they can *provide* or that they *require* from the other player.

Crucially, we know that there is only a constant number of nonzero demand vertices in Y, which can be seen to imply that the intersection of optimal solutions with Y is bounded. Thus, Alice can try all partial solutions $S_Y \subseteq Y$ of bounded size and determine what paths between Z and S_Y or between different vertices of Z she can provide for Bob. We formalize this set of paths as her *facilities*. She can furthermore determine what paths she needs to satisfy the demand of each of her vertices. Each demand vertex v gives rise to a set of required paths that may run between Z and Bob's solution or just between vertices of Z. We formalize this family of sets as Alice's *requirement*. (Note that in fact both players can offer or require various different *packings* of such paths.)

We now define Alice's requirements formally; the facilities are deferred to a full version. For notational convenience, we call a set of paths to be *v-independent* if each pair of paths is vertex-disjoint except for possibly sharing v as an endpoint. For a graph G, a vertex v,

an integer i, and two vertex subsets $A, B \subseteq V(G)$ we define a (v, i, A, B)-*constrained path packing* as a set of $i + |A|$ v-independent paths from $A \cup \{v\}$ to B in G. Herein we explicitly allow $v \notin V(G)$ if $i = 0$. We tacitly assume that, if $A \cap B \neq \emptyset$ then the paths corresponding to $A \cap B$ in the packing are of length zero, that is, they each comprise a single vertex.

Satisfying connectors indicate which of Bob's path packings are sufficient for Alice and a certain demand vertex.

▶ **Definition 16** (Satisfying connector)**.** Let H be a graph on vertex set $Y \uplus Z$, let $v \in Y$, let v have positive demand d_v, and let $S_Y \subseteq Y$ be a partial solution. A tuple (A, B, C) with $A, B, C \subseteq Z$, pairwise disjoint, is a *satisfying connector for v with respect to S_Y in H* if either $v \in S_Y$ or there is a $(v, d_v, A, B \cup S_Y)$-constrained path packing in $H - C$. The set of all satisfying connectors for v with respect to the partial solution S_Y is denoted by $\mathit{Sat}(H, Z, d_v, S_Y, v)$.

Intuitively, if Bob can send disjoint paths from B to his part of the solution and to A, possibly using vertices of C, then Alice can complete these paths to satisfy the demand of v. The requirement now formalizes the notion that each partial solution for Alice gives rise to a certain set of satisfying connectors.

▶ **Definition 17** (Requirement)**.** Let H be a graph on vertex set $Y \uplus Z$, let ϕ be a demand function on Y, and let $S_Y \subseteq Y$ be a partial solution. The *requirement* $\mathit{Req}(H, Z, \phi, S_Y)$ is the collection $\{\mathit{Sat}(H, Z, \phi(v), S_Y, v) \mid v \in D(H, \phi) \cap Y\}$.

It is not hard to observe that a partial solution S_Y in some $Y \in \mathcal{Y}$ has size at most $d^3 + d$ without loss of generality (recall that $|Y \cap D| \leq d^3$). Based on this fact, we can prove that $G[Y \cup Z]$ can be safely replaced by any graph G' that has the same set of facilities and the same family of requirements as $G[Y \cup Z]$ for any choice of S_Y with $|S_Y| \leq d^3 + d$. A smallest such graph G' has size bounded by some function of $|Z|$ since, if $S_Y \leq d^3 + d$, then the family of requirements has size bounded by some function in d. Checking whether the families of requirements of $G[Y \cup Z]$ and a candidate G' match can be done using a series of maximum flow computations that exploit the definition of satisfying connectors. This means that it is feasible to search incrementally for a representative G' of smallest (constant) size with the same requirement as $G[Y \cup Z]$. Similarly, we can ensure that the facilities are the same.

5.2 Kernelization procedure

Given an instance (G, ϕ, k) of VECTOR d-CONNECTIVITY(k) the kernelization proceeds as follows. Throughout, we refer to the current instance by (G, ϕ, k) and recall the use of $D = \{v \in V(G) \mid \phi(v) \geq 1\}$.

1. Apply Rule 1 exhaustively and then apply Rule 5 (this may return answer NO if we have more than d^2k demand vertices).
2. Apply the above described reduction rule once that may replace a subgraph $G[Y]$ by a smaller, constant-size graph.
3. Return to Step 1 if Step 2 was successful.
4. Let $W := D \cup \bigcup_{Y \in \mathcal{Y}} N[Y]$. Perform the torso operation on W in G to obtain G'. That is, carry out the following steps:
   a. Start with $G' = G[W]$.
   b. For every pair $u, v \in W$, if there is a u, v-path in G with internal vertices from $V \setminus W$ then add the edge $\{u, v\}$ to G'.
5. Return (G', ϕ', k) as the kernelized instance, where ϕ' is ϕ restricted to W.

Intuitively, the torso operation in Step 4 removes all vertices that are not in any set $Y \in \mathcal{Y}$ without affecting the sets therein. Overall, we obtain the following.

▶ **Theorem 18.** VECTOR d-CONNECTIVITY(k) *has a vertex-linear kernelization.*

6 Kernelization lower bound

In this section, we prove that VECTOR CONNECTIVITY(k) admits no polynomial kernelization unless NP $\subseteq$ coNP/poly. We give a reduction from HITTING SET(m), i.e., HITTING SET with parameter number of sets, which also makes a polynomial Turing kernelization unlikely (cf. [6]). Since demands greater than $k+1$ can be safely replaced by demand $k+1$, implying $d \leq k+1$, the lower bound applies also to parameterization by $d+k$.

▶ **Theorem 19.** VECTOR CONNECTIVITY(k) *does not admit a polynomial kernelization unless* NP $\subseteq$ coNP/poly *and the polynomial hierarchy collapses.*

Proof. We give a polynomial parameter transformation from HITTING SET(m) to VECTOR CONNECTIVITY(k), which is known to imply the claimed lower bound (cf. [1]). Let $(U, \mathcal{F}, k)$ be an instance of HITTING SET(m) with parameter $m = |\mathcal{F}|$; w.l.o.g. $k \leq m$. Let $n := |U|$. We construct a graph G on $2(k+1)m + n$ vertices that has a vector connectivity set of size at most $k' = (k+1)m + k = \mathcal{O}(m^2)$ if and only if $(U, \mathcal{F}, k)$ is YES for HITTING SET(m).

Construction. Make one vertex x_u for each element $u \in U$, and make $2(k+1)$ vertices $y_{1,F}, \ldots, y_{k+1,F}, y'_{1,F}, \ldots, y'_{k+1,F}$ for each set $F \in \mathcal{F}$. We add the following edges:

1. Add $\{y_{i,F}, y'_{i,F}\}$ for all $i \in \{1, \ldots, k+1\}$ and $F \in \mathcal{F}$.
2. Add $\{x_u, y_{i,F}\}$ for all $i \in \{1, \ldots, k+1\}$, $F \in \mathcal{F}$, and $u \in F$.
3. Make the set of all vertices $y_{i,F}$ a clique (not including any y'-vertex).

Set the demand ϕ of each $y'_{i,F}$ vertex to 2 and of each $y_{i,F}$ vertex to $(k+1)m+1$; all x-vertices have demand zero. Set the budget k' to $(k+1)m + k$. This completes the construction of an instance (G, ϕ, k'), which can be easily performed in polynomial time.

Correctness. Assume first that (G, ϕ, k') is YES and let S a vector connectivity set of size at most k'. Note that S must contain all vertices $y'_{i,F}$ since they have demand of 2 but only one neighbor (namely $y_{i,F}$). This accounts for $(k+1)m$ vertices in S; there are at most k further vertices in S. Let T contain exactly those elements $u \in U$ such that $x_u \in S$; thus $|T| \leq k$. We claim that T is a hitting set for $\mathcal{F}$. Let $F \in \mathcal{F}$ and assume that $T \cap F = \emptyset$. It follows that S contains no vertex x_u with $u \in F$. Since at most k vertices in S are not y'-vertices, we can choose $i \in \{1, \ldots, k+1\}$ such that S does not contain $y_{i,F}$. Consider the set C consisting of all y-vertices other than $y_{i,F}$ as well as the vertex $y'_{i,F}$. In $G - C$ we find a connected component containing $y_{i,F}$ and all x_u with $u \in F$ but no further vertices. Crucially, all other neighbors of $y_{i,F}$ are $y'_{i,F}$ and all y-vertices, and x-vertices only have y-vertices as neighbors. By assumption S contains no vertex of this connected component. This yields a contradiction cause C is of size $(k+1)m$ and separates $y_{i,F}$ from S, but since S is a solution with $y_{i,F} \notin S$ there should be $(k+1)m+1$ disjoint paths from $y_{i,F}$ to S. Thus, S must contain some x_u with $u \in F$, and then $T \cap F \neq \emptyset$.

Now, assume that $(U, \mathcal{F}, k)$ is YES for HITTING SET(m) and let T a hitting set of size at most k for $\mathcal{F}$. We create a vector connectivity set S by selecting all x_u with $u \in T$ as well as all y'-vertices; thus $|S| \leq k' = (k+1)m + k$. Clearly, this satisfies all y'-vertices. Consider any vertex $y_{i,F}$ and recall that its demand is $\phi(y_{i,F}) = (k+1)m + 1$. We know that S contains at least one vertex x_u with $u \in F$ that is adjacent to $y_{i,F}$. Thus, we can find the required $(k+1)m+1$ disjoint paths from $y_{i,F}$ to S:

- We have one path $(y_{i,F}, y'_{i,F})$ and one path $(y_{i,F}, x_u)$.
- For all $(j, F') \neq (i, F)$ we get one path $(y_{i,F}, y_{j,F}, y'_{j,F})$; we get $(k+1)m - 1$ paths total.

It follows that (G, ϕ, k') is YES for VECTOR CONNECTIVITY(k).

We have given a polynomial parameter transformation from HITTING SET(m), which is known not to admit a polynomial kernelization unless NP $\subseteq$ coNP/poly [4] (see also [6]). This is known to imply the same lower bound for VECTOR CONNECTIVITY(k) [1]. ◀

7 Conclusion

We have presented kernelization and approximation results for VECTOR CONNECTIVITY and VECTOR d-CONNECTIVITY. An important ingredient of our results is a reduction rule that reduces the number of vertices with nonzero demand to at most $d^2\mathsf{opt}$ (or, similarly, to at most $\mathsf{opt}^3 + \mathsf{opt}$ or $k^3 + k$). From this, one directly gets approximation algorithms with ratios d^2 and $(\mathsf{opt}^2 + 1)$; we improved these to factors d and opt, respectively, by a local-ratio type algorithm. Recall that VECTOR d-CONNECTIVITY is APX-hard already for $d = 4$ [3].

On the kernelization side we show that VECTOR CONNECTIVITY(k) does not admit a polynomial kernelization unless NP $\subseteq$ coNP/poly. Since demands greater than $k + 1$ can be safely replaced by demand $k + 1$ (because they cannot be fulfilled without putting the vertex into the solution) the lower bound extends also to parameter $k + d$. For VECTOR d-CONNECTIVITY(k), where d is a problem-specific constant, we give an explicit vertex-linear kernelization with at most $f(d) \cdot k = \mathcal{O}(k)$ vertices; the computable function $f(d)$ is superpolynomial in d, which is necessary (unless NP $\subseteq$ coNP/poly) due to the lower bound for $d + k$.

Finally, the reduction to $\ell \leq k^3 + k$ nonzero demand vertices allows an alternative proof for fixed-parameter tractability: We give a randomized FPT-algorithm for VECTOR CONNECTIVITY(k) that finds a solution by seeking a set of size k that is simultaneously independent in each of ℓ linear matroids; for this we use an algorithm of Marx [8] for linear matroid intersection, which is fixed-parameter tractable in $k + \ell = \mathcal{O}(k^3)$.

Acknowledgments. The authors are grateful to anonymous reviewers for suggesting a simplified definition for signatures and pointing out the non-constructive kernelization argument that can be obtained from the work of Fomin et al. [5].

References

1 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM J. Discrete Math.*, 28(1):277–305, 2014.

2 Endre Boros, Pinar Heggernes, Pim van 't Hof, and Martin Milanič. Vector connectivity in graphs. *Networks*, 63(4):277–285, 2014.

3 Ferdinando Cicalese, Martin Milanič, and Romeo Rizzi. On the complexity of the vector connectivity problem. *Theor. Comput. Sci.*, 591:60–71, 2015.

4 Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Kernelization lower bounds through colors and IDs. *ACM T. Alg.*, 11(2):13:1–13:20, 2014.

5 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Linear kernels for (connected) dominating set on graphs with excluded topological subgraphs. In *Proc. 30th STACS*, pages 92–103. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

6 Danny Hermelin, Stefan Kratsch, Karolina Soltys, Magnus Wahlström, and Xi Wu. A completeness theory for polynomial (Turing) kernelization. *Algorithmica*, 71(3):702–730, 2015.

7 Stasys Jukna. *Extremal combinatorics - with applications in computer science.* Texts in theoretical computer science. Springer, 2001.

8 Dàniel Marx. A parameterized view on matroid optimization problems. *Theor. Comput. Sci.*, 410(44):4471–4479, 2009.

B-Chromatic Number: Beyond NP-Hardness*

Fahad Panolan[1], Geevarghese Philip[2], and Saket Saurabh[1,3]

1 **Institute of Mathematical Sciences, Chennai, India**
{fahad|saket}@imsc.res.in
2 **Chennai Mathematical Institute, India**
gphilip@cmi.ac.in
3 **University of Bergen, Norway**

Abstract

The b-chromatic number of a graph G, $\chi_b(G)$, is the largest integer k such that G has a k-vertex coloring with the property that each color class has a vertex which is adjacent to at least one vertex in each of the other color classes. In the b-Chromatic Number problem, the objective is to decide whether $\chi_b(G) \geq k$. Testing whether $\chi_b(G) = \Delta(G) + 1$, where $\Delta(G)$ is the maximum degree of a graph, itself is NP-complete even for connected bipartite graphs (Kratochvíl, Tuza and Voigt, WG 2002). In this paper we study b-Chromatic Number in the realm of parameterized complexity and exact exponential time algorithms. We show that b-Chromatic Number is W[1]-hard when parameterized by k, resolving the open question posed by Havet and Sampaio (Algorithmica 2013). When $k = \Delta(G) + 1$, we design an algorithm for b-Chromatic Number running in time $2^{\mathcal{O}(k^2 \log k)} n^{\mathcal{O}(1)}$. Finally, we show that b-Chromatic Number for an n-vertex graph can be solved in time $\mathcal{O}(3^n n^4 \log n)$.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases b-chromatic number, exact algorithm, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.389

1 Introduction

Graph coloring (proper vertex coloring), is an assignment of colors to the vertices of a graph such that no edge connects two identically colored vertices. In other words graph coloring is a partition of vertex set into independent sets. A proper vertex coloring using k colors is called a *k-vertex coloring*. The least number of colors required for a proper vertex coloring of a graph G is called the *chromatic number* of G. The most common question about graph coloring is–"what is the chromatic number of a graph". This question has got lots of attention in graph theory and algorithms. The study of graph coloring leads to the four color theorem in planar graphs by Appel and Haken [1], study of chromatic polynomial introduced by Birkhoff, which was generalised to the Tutte polynomial by Tutte, etc. Graph coloring has been studied as an algorithmic problem since the early 1970s. The chromatic number problem is one of Karp's 21 NP-complete problems from 1972 [11]. An exact algorithm to compute the chromatic number of a graph dates back to 1976. Lawler [13] gave an algorithm for finding chromatic number running in time $2.4423^n n^{\mathcal{O}(1)}$. Finally, after 30 years, using the principle of inclusion-exclusion Björklund et al. [2] gave an algorithm for chromatic number

* Supported by the European Research Council (ERC) via grants Rigorous Theory of Preprocessing, reference 267959 and PARAPPROX, reference 306992; and by the Department of Science and Technology (DST), Government of India, the German Federal Ministry of Education and Research (BMBF), and the Max Planck Society (MPG), via the Indo-German Max Planck Center for Computer Science (IMPECS).

problem running in time $2^n n^{\mathcal{O}(1)}$. This is still the fastest known exact algorithm to compute the chromatic number of a graph.

Not only finding chromatic number but also different variations of graph coloring has been studied in the literature. A *complete coloring* of a graph G is a proper vertex coloring such that no two color classes together form an independent set. The parameter *achromatic number* of a graph G is the largest integer k such that there is a complete coloring of G using k colors. Irving and Manlove [10] introduced *b-chromatic number*, another parameter related to graph coloring. The *b-chromatic number* of a graph G, denoted by $\chi_b(G)$, is the largest integer k such that G has a k-vertex coloring with the property that each color class has a vertex which is adjacent to at least one vertex in each of the other color classes. Such a coloring is called a *b-coloring*. Irving and Manlove showed that determining b-chromatic number is NP-complete for general graphs, but polynomial time solvable for trees [10]. From the definition of b-chromatic number it is clear that $\chi_b(G) \leq \Delta(G) + 1$, where $\Delta(G)$ is the maximum degree of the graph G. Kratochvíl et al. [12] showed that determining whether $\chi_b(G) = \Delta(G)+1$ is NP-hard even for connected bipartite graphs. Havet et al. [8] showed that b-chromatic number can be computed in polynomial time for split graphs and it is NP-hard for connected chordal graphs. Regarding approximation algorithms for the problem, Galcík et al. [7] showed that b-chromatic number of an n-vertex graph can not be approximated within a factor $n^{1/4-\epsilon}$ for any constant $\epsilon > 0$, in polynomial time, unless P =NP.

In this work we address the algorithmic question of b-chromatic number in the realm of parameterized complexity and exact exponential time algorithms.

b-Chromatic Number	**Parameter:** k
Input: An n-vertex graph G and an integer k	
Question: Is the b-chromatic number of G is at least k	

For a detailed overview of parameterized complexity reader is referred to monographs [5, 4]. In the parameterized complexity framework, the b-Chromatic Number problem is studied with a dual parameter by Havet et al. [9]. In particular, they show that one can decide whether $\chi_b(G) \geq n - k$ in time $2^{\mathcal{O}(k \log k)} n^{\mathcal{O}(1)}$ and asked the question whether b-Chromatic Number is FPT when parameterized by k. Recently, Effantin et al. [6] studied a relaxed version of b-coloring and repeated the question about the parameterized complexity of b-Chromatic Number. In this work we answer this question negatively, by showing that b-Chromatic Number is W[1]-hard. But, when $k = \Delta(G)+1$, we design an algorithm for b-Chromatic Number running in time $2^{\mathcal{O}(k^2 \log k)} n^{\mathcal{O}(1)}$. Finally we show that b-Chromatic Number for an n-vertex graph can be solved in time $\mathcal{O}(3^n n^4 \log n)$. After the results in this article were presented at IPEC, Manfred Cochefert informed us that he had derived, in his PhD thesis [3] a $3^n n^{\mathcal{O}(1)}$ time algorithm for b-Chromatic Number using principle of inclusion-exclusion.

Our methods. To show b-Chromatic Number is W[1]-hard, when parameterized by k, we give an FPT-reduction from Multi Colored Independent Set, which is very well known to be W[1]-hard [4]. When $k = \Delta(G) + 1$, to get an FPT algorithm for b-Chromatic Number, we first show that it is enough to find $C \subseteq V(G)$ such that $\chi_b(G[C]) = k$ (we call such a subset C as b-chromatic core of order k). Then we give a polynomial kernel for the problem of finding b-chromatic core of order $\Delta(G) + 1$, which leads to an FPT algorithm for b-Chromatic Number when $k = \Delta(G) + 1$. For the exact exponential time algorithm for b-Chromatic Number, we reduce the problem to many instances of single variate polynomial multiplication of degree 2^n.

2 Preliminaries

We use "graph" to denote simple graphs without self-loops, directions, or labels. We use $V(G)$, $E(G)$ and $\Delta(G)$, respectively, to denote the vertex set, edge set and maximum degree of a graph G. We also use $G = (V, E)$ to denote a graph G on vertex set V and edge set E. For $v, u \in V(G)$ and $V' \subseteq V(G)$, we use $G[V']$ to denote the subgraph of G induced on V', $N[v] = \{u \ : \ (v, u) \in E(G)\} \cup \{v\}$ and $d(u, v)$ is the shortest distance between u and v. For a graph G and a b-coloring of G with color classes $C_1, \ldots, C_k$, we say a vertex $v \in C_i$ is a *dominating* vertex for the color class C_i if v is adjacent to a vertex in C_j for each $j \neq i$.

We use $[n]$ to denote the set $\{1, 2, \ldots, n\}$. We use $\uplus$ to denote the disjoint union of sets: for any two sets A, B, the set $A \uplus B$ is defined only if $(A \cap B) = \emptyset$, and in this case $(A \uplus B) = (A \cup B)$. We assume that $\uplus$ associates to the left; that is, we write $\biguplus_{1 \leq i \leq n} A_i = A_1 \uplus A_2 \uplus A_3 \cdots \uplus A_n$ to mean $(\cdots((A_1 \uplus A_2) \uplus A_3) \cdots \uplus A_n)$. Further, every use of $\uplus$ in an expression carries with it the implicit assertion that the two sets involved are disjoint.

A *binary vector* is a finite sequence of bits, and its *width* is the number of bits in the sequence. If A, B are binary vectors, then by $A + B$ we mean the *integer* $val(A) + val(B)$ where for a binary vector X the expression $val(X)$ denotes the integer of which X is a binary representation. Let $U = \{u_1, u_2, \ldots, u_n\}$ be a set of cardinality n, and let $S \subseteq U$. The *characteristic vector* $\chi(S)$ of S with respect to U is the binary vector with $|U| = n$ bits whose ℓ^{th} bit, for $1 \leq \ell \leq n$, is 1 if element u_ℓ belongs to set S, and 0 otherwise. We use $\mathbb{N}$ to denote the set of non-negative integers. The *Hamming weight* $\mathcal{H}(r)$ of a binary vector r is the number of 1s in r. For a finite set U, a subset $S \subseteq U$, and the characteristic vector $\chi(S)$ of S with respect to U, observe that $\mathcal{H}(\chi(S)) = |S|$. We define the Hamming weight of $n \in \mathbb{N}$ to be the number $\mathcal{H}(n)$ of the number of 1s in a binary representation of n. Note that an integer $n \in \mathbb{N}$ does not have a *unique* binary representation, since we can pad any such representation with zeroes on the left without changing its numerical value. We call the total number of bits in a binary representation r of n the *width* of r.

▶ **Lemma 1** ($\star$[1]). *Let S_1, S_2 be two disjoint subsets of a set $U = \{u_1, u_2, \ldots, u_n\}$, and let $S = S_1 \uplus S_2$. Then*

1. $\chi(S) = \chi(S_1) + \chi(S_2)$
2. $\chi(S_1) + \chi(S_2)$ *has a binary representation of width n*
3. $\mathcal{H}(\chi(S)) = \mathcal{H}(\chi(S_1)) + \mathcal{H}(\chi(S_2))$

We make extensive use of single-variate polynomials with integer coefficients. Let $P = \sum_{i=0}^{n} a_i x^i$ be such a polynomial. We say that polynomial P *contains* monomial x^i, or that monomial x^i is *present in* polynomial P, if the coefficient a_i of x_i is not zero. We say that the polynomial $P' = \sum_{i=0}^{n} b_i x^i$ is the *representative polynomial* of P if $b_i = 1$ whenever $a_i \neq 0$ and $b_i = 0$ otherwise. That is, the representative polynomial remembers just the *degrees* of the monomials which are present in P, and forgets their coefficients. For an $h \in \mathbb{N}$, we define the *Hamming projection* of polynomial P to h to be $\mathcal{H}_h(P) = \sum_{i=0}^{n} b_i x^i$ where $b_i = a_i$ if $\mathcal{H}(i) = h$ and 0 otherwise. That is, $\mathcal{H}_h(P)$ is the sum of all those monomials in P whose *degrees* have Hamming weight h. To obtain the stated running time for our exponential-time algorithm, we make use of the fast algorithm for multiplying polynomials which is based on the Fast Fourier Transform.

[1] Proofs of results marked with a $\star$ are deferred to the full version of the paper.

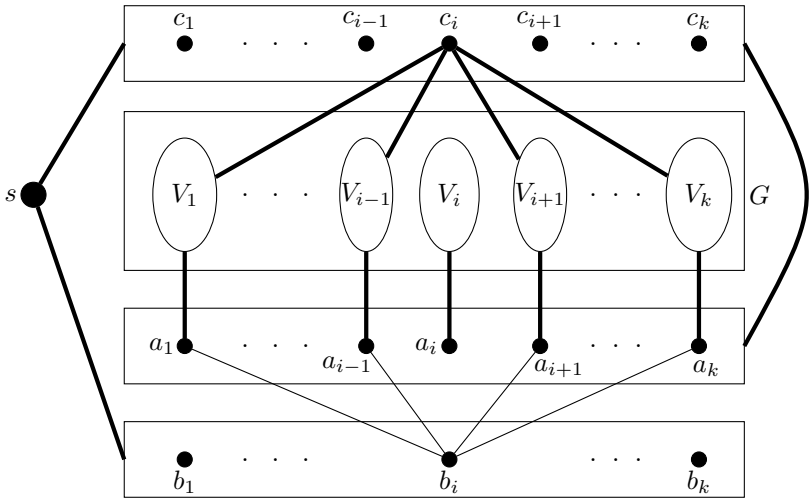


Figure 1 The graph G' constructed from the input instance $G = (V_1 \uplus \ldots \uplus V_k, E)$ of Multi-Colored Independent Set. The thick edges represent all possible edges between two corresponding sets of vertices.

▶ **Lemma 2** ([14]). *Two polynomials of degree at most n over any commutative ring $\mathcal{R}$ can be multiplied using $\mathcal{O}(n \log n \log\log n)$ additions and multiplications in $\mathcal{R}$.*

3 Hardness

In this section we show that b-Chromatic Number is W[1]-hard by giving an FPT-reduction from Multi-Colored Independent Set.

Multi-Colored Independent Set	**Parameter:** k
Input: A k-partite graph G with its k-partition $V_1 \uplus \ldots \uplus V_k$ of $V(G)$	
Question: Is there an independent set of size k containing one vertex from each V_i?	

▶ **Theorem 3.** *There is a polynomial time algorithm that given an instance $G = (V_1 \uplus \ldots \uplus V_k, E)$ of* Multi-Colored Independent Set*, constructs an instance $(G', 2k+1)$ of* b-Chromatic Number *such that G is a* Yes *instance of* Multi-Colored Independent Set *if and only if $(G', 2k+1)$ is a* Yes *instance of* b-Chromatic Number*.*

Proof. Let G be an instance of Multi-Colored Independent Set, with its k-partition $V_1 \uplus \ldots \uplus V_k$.

Construction. We construct a graph G' from G as follows. The vertex set of G', $V(G') = V(G) \cup A \cup B \cup C \cup \{s\}$, where $A = \{a_1, \ldots, a_k\}, B = \{b_1, \ldots, b_k\}$ and $C = \{c_1, \ldots, c_k\}$. The edge set $E(G')$ contains $E(G)$ and the following sets of new edges (see Figure 1).

- $\{(a_i, a_j) \mid i \neq j\} \cup \{(c_i, c_j) \mid i \neq j\}$ (i.e A and C forms cliques),
- $\{(a_i, c_j) \mid 1 \leq i, j \leq k\}$ (i.e $A \cup C$ forms a clique),
- $\{(a_i, b_j) \mid i \neq j\}$,
- $\{(a_i, v) \mid v \in V_i\} \cup \{(c_i, v) \mid v \in V(G) \setminus V_i\}$,
- $\{(s, b_i), (s, c_i) \mid 1 \leq i \leq k\}$.

Completeness. Suppose G is an YES instance of MULTI-COLORED INDEPENDENT SET. Let $I = \{v_1, \ldots, v_k\}$ be an independent set such that $v_i \in V_i$ for all $i \in [k]$. Now we give a b-coloring of G' using $2k+1$ colors as follows. We define $2k+1$ color classes – $C_0, C_1, \ldots, C_{2k}$. The color class $C_0 = I \cup \{s\}$. For all $1 \leq i \leq k$, $C_i = (V_i \cup \{c_i\}) \setminus \{v_i\}$. For all $k+1 \leq i \leq 2k$, $C_i = \{a_i, b_i\}$. Note that for all $0 \leq i \leq 2k$, C_i is an independent set in G' and $C_0 \uplus C_1 \uplus \ldots \uplus C_{2k} = V(G')$. Now we show that each color class has a dominating vertex. For color class C_0, the vertex s is a dominating vertex, because s is adjacent to all the vertices in the set $B \cup C$. For each $1 \leq i \leq k$, the vertex c_i is the dominating vertex of the color class C_i, because it is adjacent to the vertices $A \cup \{s\}$ and $C \setminus \{c_i\}$. For each $k+1 \leq i \leq 2k$, a_i is the dominating vertex for the color class C_i, because it is adjacent to all the vertices in $C \cup (A \setminus \{a_i\})$ and the vertex v_i in the color class C_0.

Soundness. Let $(G', 2k+1)$ is an YES instance of b-CHROMATIC NUMBER. Let ϕ be a b-coloring for the graph G' using at least $2k+1$ colors. Since $A \cup C$ is a clique, all the vertices in $A \cup C$ are colored differently by ϕ. For $1 \leq i \leq k$, let C_i be the the color class which contains the vertex c_i and for each $k+1 \leq i \leq 2k$, let C_i be the color class which contains the vertex a_i. Let C_0 be an arbitrary color class other that $C_1, \ldots, C_{2k}$ in the coloring ϕ.

First note that since the degree of any vertex in B is k, no vertex in B can be a dominating vertex for any color class. Now consider the following claim.

▶ **Claim 1.** *No vertex $u \in V(G)$ can be a dominating vertex for any of the color classes $C_0, C_{k+1}, C_{k+2}, \ldots, C_{2k}$.*

Proof. Suppose $u \in V(G)$ is a dominating vertex for the color class C_j for some $j \in \{0, k+1, k+2, \ldots, 2k\}$. Let $u \in V_i$ for some $1 \leq i \leq k$. Consider the color class C_i. Note that $c_i \in C_i$. This implies that $C_i \subseteq V_i \cup B$, because the non-neighborhood of c_i is $V_i \cup B$. But since $u \in V_i$, u is not adjacent to any vertex in $V_i \cup B$. This contradicts that u is a dominating vertex for C_j. ◀

Since C_0 is disjoint from $A \cup C$ and by Claim 1, we can conclude that s is the dominating vertex for C_0. Also note that for any $k+1 \leq i \leq 2k$, C_i is disjoint from $(A \cup C \cup \{s\}) \setminus \{a_i\}$. By Claim 1, this implies that a_i is the dominating vertex for the class C_i. Since a_i is a dominating vertex for the class C_i for each $k+1 \leq i \leq 2k$, there is a vertex v in C_0 such that $(a_i, v) \in E(G')$. Since $C_0 \cap (A \cup B \cup C) = \emptyset$ (because $s \in C_0$), we have that $v \in V_{i-k}$. This implies that for each $k+1 \leq i \leq 2k$, there is a vertex $v \in V_{i-k}$ such that $v \in C_0$. This implies that G has an independent set of size k containing one vertex from each V_j for $1 \leq j \leq k$. This completes the proof of the lemma. ◀

4 FPT Algorithm for deciding whether $\chi_b(G) = \Delta(G) + 1$

In this section we design a parameterized algorithm for b-CHROMATIC NUMBER to decide whether $\chi_b(G) = \Delta(G) + 1$. For this section we set $k = \Delta(G) + 1$. Towards this we define a notion of b-*chromatic-core*. Given a graph G, and a positive integer ℓ, a set $C \subseteq V(G)$ is called a b-*chromatic-core of order* ℓ if $\chi_b(G[C]) \geq \ell$. Observe that a minimal set such that $\chi_b(G[C]) \geq \ell$ has size upper bounded by ℓ^2. We start by showing that for the case, when we want to test whether $\chi_b(G) = \Delta(G) + 1 = k$, it is sufficient to find a b-chromatic-core of order k.

▶ **Lemma 4.** *Let G be a graph. Then $\chi_b(G) = k$ if and only if G has a b-chromatic-core of order k. Here $k = \Delta(G) + 1$.*

Proof. For the forward direction let $\chi_b(G) = k$. Then $V(G)$ is a b-chromatic-core of order k.

For the reverse direction assume that G has a b-chromatic-core C of order k. By the definition of b-chromatic-core we have that $\chi_b(G[C]) \geq k$. Since $\chi_b(G[C]) \leq \Delta(G) + 1$ we have that $\chi_b(G[C]) = k$. Let $C_1, \ldots, C_k$ be the partition of C witnessing the fact that $\chi_b(G[C]) = k$. Now we will show that we can extend this partition to the vertex set of G. Let $w_1, \ldots, w_q$ denote the vertices of $V(G) \setminus C$. We iteratively go through the vertices in $V(G) \setminus C$ and try to place it in the already existing partition. Suppose at some stage we have taken care of vertices until say w_i. For w_{i+1} we do as follows. Since the degree of every vertex is upper bounded by Δ we have that there is a partition C_j such that w_{i+1} does not have any neighbor in C_j. We place $w_{i+1} \in C_j$. That is, $C_j := C_j \cup \{w_{i+1}\}$. Observe that the placement preserves the fact that C_j after the addition of the vertex remains independent. This proves the lemma. ◀

Lemma 4 allows us to look for b-chromatic-core of order k for G. Towards this we define the following reduction rule.

▶ **Reduction Rule 1.** *Let (G, k) be an instance to* b-Chromatic Number *and v be a vertex such that every vertex $w \in N[v]$ has degree at most $k - 2$. Then $(G \setminus \{v\}, k)$.*

▶ **Lemma 5 ($\star$).** *Reduction Rule 1 is safe. That is, (G, k) is a* Yes *instance to* b-Chromatic Number *if and only if $(G' = G \setminus \{v\}, k)$ is a* Yes *instance to* b-Chromatic Number.

▶ **Theorem 6.** *Let (G, k) be an instance to* b-Chromatic Number *such that $k = \Delta(G) + 1$. Then there is an algorithm that decides whether $\chi_b(G) = k$, in time $2^{\mathcal{O}(k^2 \log k)} + n^{\mathcal{O}(1)}$.*

Proof. We first apply Reduction Rule 1 exhaustively. Then, we greedily find a maximal set S such that (a) degree of every vertex is equal to $k - 1$ and (b) for any pair of vertices $v, w \in S$, we have that $d(v, w) \geq 4$. We have two cases either $|S| \geq k$ or $|S| < k$. In the first case we can show that $\chi_b(G) = k$ in polynomial time and in the later case we will bound the number of vertices of G by a polynomial function of k.

Case I: $|S| \geq k$. In this case we will show that S and its neighbors form b-chromatic-core of order k for G. Let $C = \{v_1, \ldots, v_k\}$ be an arbitrary subset of size k of S. For every $v_i \in C$ let $W_i = N(v_i)$. Let the vertices of W_i be denoted by $\{w_1^i, \ldots, w_{i-1}^i, w_{i+1}^i, w_k^i\}$. Observe that since for any pair of vertices $v, w \in S$, $d(v, w) \geq 4$, we have that $W_i \cap W_j = \emptyset$ and there are no edges between W_i and W_j for $i \neq j$. Now we make color class I_i, $i \in \{1, \ldots, k\}$, as follows: $I_i = \{w_i^j \mid j \neq i\} \cup \{v_i\}$. Observe that by construction every vertex v_i has neighbor in every color class except I_i and thus $C \cup W$ forms a b-chromatic-core of order k for G. Given the partition of $C \cup W$ we can find a b-chromatic partition of G in polynomial time using the procedure described in Lemma 4.

Case II: $|S| < k$. In this case we claim that for every vertex $v \in V(G) \setminus S$, there exists a vertex $u \in S$ such that $d(u, v) \leq 4$. First of all notice that either the degree of v is $k - 1$ or has a neighbor w with degree $k - 1$. The last assertion follows from the fact that we can not apply Reduction Rule 1 on G. Suppose the degree of v is $k - 1$ then since the greedy algorithm did not pick $v \in S$, we have that there exists a vertex $u \in S$ such that $d(u, v) \leq 3$. In the case when w has degree $k - 1$, we have that there exists a vertex $u \in S$ such that $d(u, w) \leq 3$ and thus $d(u, v) \leq 4$. This implies that every vertex in $V(G)$ can be reached from a vertex in S by a path of length at most 4. Since the maximum degree of this graph is at most $k - 1$, we have that $|V(G)| = 5k^5 = \mathcal{O}(k^5)$. Thus, to test whether $\chi_b(G) = k$ we

guess the b-chromatic-core of order k. We know that there exists one of size at most k^2. Thus, this results in

$$\binom{5k^5}{k^2} \leq 2^{\mathcal{O}(k^2 \log k)}$$

guesses. Given C we can test whether $\chi_b(G[C]) = k$ in time $2^{\mathcal{O}(|C|)} = 2^{\mathcal{O}(k^2)}$ (see the exact algorithm for the mentioned running time). Even the brute force partition into k parts will led to an algorithm with running time $2^{\mathcal{O}(k^2 \log k)}$. This concludes the proof. ◀

5 Exact Algorithm

In this section we show that given a graph G on n vertices as input, we can find the b-chromatic number of G in running time which is single-exponential in n, modulo polynomial factors:

▶ **Theorem 7.** *There is an algorithm which, given a graph G on n vertices as input, finds the b-chromatic number of G in $\mathcal{O}(3^n n^4 \log n)$ time.*

Our algorithm works by checking, for $k = n, (n-1), \ldots, 1$ in this order, whether G has a b-coloring with k colors. It outputs the first (and so, the largest) value of k for which the check returns Yes.

▶ **Theorem 8.** *There is an algorithm which, given a graph G on n vertices and an integer $1 \leq k \leq n$ as input, checks if G has a b-coloring with k colors in $\mathcal{O}(\binom{n}{k} 2^{(n-k)} n^4 \log n)$ time.*

Given Theorem 8, the proof of Theorem 7 is immediate:

Proof of Theorem 7. Since our algorithm for Theorem 7 consists of invoking the algorithm of Theorem 8 once for each $k \in \{1, 2, \ldots, n\}$, we get that the former algorithm runs in time

$$\sum_{0 \leq k \leq n} \mathcal{O}(\binom{n}{k} 2^{(n-k)} n^4 \log n) = \mathcal{O}(n^4 \log n \sum_{0 \leq k \leq n} \binom{n}{k} 2^{(n-k)}) = \mathcal{O}(3^n n^4 \log n),$$

where we get the simplified form from the Binomial Theorem. ◀

Observe that our goal in Theorem 8 is to check whether we can partition the vertex set of G into exactly k non-empty parts $V_1, V_2, \ldots, V_k$ such that

- each set V_i is an independent set in G, and
- for each $1 \leq i \leq k$ there is a vertex $v_i \in V_i$ which has a neighbour in each of the other sets V_j ; $j \neq i$.

Such a partition is a b-coloring of G with k colors, and we call such a set of k vertices $\{v_1, v_2, \ldots, v_k\}$ a *dominator set* for the b-coloring $V_1, V_2, \ldots, V_k$. Our algorithm for Theorem 8 checks, for each vertex subset $\{v_1, v_2, \ldots, v_k\}$ of G of size k, whether there is a b-coloring of G with k colors for which $\{v_1, v_2, \ldots, v_k\}$ is a dominator set.

▶ **Lemma 9.** *Given a graph G on n vertices and a vertex subset $D = \{v_1, v_2, \ldots, v_k\}$ of G of size k, we can find whether graph G has a b-coloring with k colors for which D is a dominator set, in $\mathcal{O}(2^{(n-k)} n^4 \log n)$ time.*

Note that Theorem 8 follows directly from Lemma 9. In the next subsection we describe an algorithm of the kind specified in Lemma 9. We then prove its correctness (subsection 5.2) and show that it runs within the stated time bounds (subsection 5.3).

5.1 The Algorithm

We describe an algorithm which, given a graph G on n vertices and a vertex subset $D = \{v_1, v_2, \ldots, v_k\}$ of G of size k, finds whether graph G has a b-coloring with k colors for which D is a dominator set. Let $\mathcal{I}$ denote the set of all vertex subsets of G which are independent sets in G. Let $V' = V(G) \setminus D$ be the set of all vertices of graph G which are *not* part of the candidate dominator set D. We label the vertices of V' as $V' = \{u_1, u_2, \ldots, u_{n-k}\}$. Let x be an indeterminate.

For each pair of distinct indices $i, j\ ;\ 1 \leq i \neq j \leq k$, let $\mathcal{S}_{ij}$ denote the set of subsets X of V' such that $(X \cup \{v_i\})$ is an independent set in G, and $(X \cup \{v_i, v_j\})$ is *not* an independent set in G:

$$\mathcal{S}_{ij} = \{X \subseteq V'\ ;\ (X \cup \{v_i\}) \in \mathcal{I} \text{ and } \forall 1 \leq j \neq i \leq k : (X \cup \{v_i, v_j\}) \notin \mathcal{I}\} \tag{1}$$

For each index $i\ ;\ 1 \leq i \leq k$, let $\mathcal{T}_i$ denote the intersection, over all $j \neq i$, of the sets $\mathcal{S}_{ij}$:

$$\mathcal{T}_i = \bigcap_{1 \leq j \neq i \leq k} \mathcal{S}_{ij} \tag{2}$$

Note that $\mathcal{T}_i$ is the set of all independent sets in V' which could *potentially* form a color class together with vertex v_i in a b-coloring of interest to us. Indeed, *any* b-coloring of G of the specified kind will consist—apart from the vertices of D—of a pairwise disjoint collection of k independent sets, one from each set $\mathcal{T}_i\ ;\ 1 \leq i \leq k$, such that their union makes up all of V'. Our algorithm looks for such a collection of independent sets, one from each set $\mathcal{T}_i$.

For each vertex $v_i \in D$ we construct the set $\mathcal{T}_i$. We then use $\mathcal{T}_i$ to construct a polynomial $\mathcal{P}_i$ in x for each vertex $v_i \in D$, in the following manner: We initialize $\mathcal{P}_i$ to zero. For each set $X \in \mathcal{T}_i$ we compute the characteristic vector $\chi(X)$ of X with respect to the set V'. We then add the monomial $x^{\chi(X)}$ to the polynomial $\mathcal{P}_i$:

$$\mathcal{P}_i = \sum_{X \in \mathcal{T}_i} x^{\chi(X)} \tag{3}$$

We now compute a sequence of polynomials $\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_k$. We set $\mathcal{Q}_1 = \mathcal{P}_1$. Now for each $2 \leq \ell \leq k$ we compute the polynomial $\mathcal{Q}_\ell$ as follows. We initialize $\mathcal{Q}_\ell$ to zero. For each pair of integers $i, j \geq 0\ ;\ i + j \leq (n - k)$, we

- Compute the polynomials $Q_i = \mathcal{H}_i(\mathcal{Q}_{(\ell-1)})$ and $P_j = \mathcal{H}_j(\mathcal{P}_{(\ell)})$, and their product $R'_{ij} = Q_i \times P_j$;
- Compute the representative polynomial R''_{ij} of R'_{ij}.
- Compute the Hamming projection $R_{ij} = \mathcal{H}_{(i+j)}(R''_{ij})$;
- Set $\mathcal{Q}'_\ell := \mathcal{Q}_\ell + R_{ij}$.
- Set $\mathcal{Q}_\ell$ to be the representative polynomial of $\mathcal{Q}'_\ell$.

▶ Remark. Note that this construction ensures that every nonzero monomial in each polynomial $\mathcal{Q}_i\ ;\ 1 \leq i \leq k$ has the form x^d for some $d \in \mathbb{N}$. That is, no monomial has a coefficient greater than 1 in any of these polynomials.

Our algorithm returns YES if the polynomial $\mathcal{Q}_k$ contains at least one monomial whose degree has Hamming weight $(n - k)$, and NO otherwise. This completes the description of the algorithm.

5.2 Correctness

We now prove that the algorithm of the previous section indeed works exactly as specified in the statement of Lemma 9. We prove this in two parts; first we show that the algorithm is complete (Lemma 10), and then we show that it is sound (Lemma 12).

▶ **Lemma 10.** *If a graph G on n vertices has a b-coloring with k colors for which $D = \{v_1, v_2, \ldots, v_k\} \subseteq V(G)$ is a dominator set, then the algorithm of subsection 5.1 returns* YES *on input (G, D).*

Proof. Suppose graph G has a b-coloring $V_1, V_2, \ldots, V_k$ with k colors for which the set $D = \{v_1, v_2, \ldots, v_k\}$ is a dominator set. Let $V' = V(G) \setminus D$ and for $1 \leq i \leq k$, let $X_i = V_i \setminus \{v_i\}$. Let $\chi(X)$ be the characteristic vector of a subset $X \subseteq V'$ with respect to the set V'. For each $1 \leq i \leq k$ let $m_i = x^{\chi(X_i)}$. It is not difficult to see that for each $1 \leq i \leq k$, the set X_i contributes the monomial m_i to the polynomial $\mathcal{P}_i$ computed by the algorithm.

▶ **Claim 2 ($\star$).** *For each $1 \leq i \leq k$, the polynomial $\mathcal{P}_i$ constructed by the algorithm contains the monomial $m_i = x^{\chi(X_i)}$.*

Our method of computing the polynomials $\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_k$ has two desirable implications: (i) the final polynomial $\mathcal{Q}_k$ computed by the algorithm contains the product of all the m_is as a monomial, and (ii) the degree of this monomial has Hamming weight exactly $(n-k)$. To see this, consider first some properties satisfied by all the polynomials $\mathcal{Q}_\ell$:

▶ **Claim 3.** *For each $1 \leq \ell \leq k$, the following hold:*

1. *Let d be the degree of the monomial $m = \prod_{1 \leq i \leq \ell} m_i$. Then*
   a. *$d \leq 2^{(n-k)}$, and hence d has a binary representation r of width $(n-k)$.*
   b. *Let $S = \biguplus_{1 \leq i \leq \ell} X_i$. Then $\chi(S) = r$.*
2. *The polynomial $\mathcal{Q}_\ell$ contains the monomial m.*

Proof of Claim 3. We prove the claim by induction on ℓ.

Base case: $\ell = 1$.

1. Here $m = m_1 = x^{\chi(X_1)}$.
   (a) Since $X_1 \subseteq V'$, $|V'| = (n-k)$, and the degree d of m_1 is $\chi(X_1)$, we have that $d \leq 2^{(n-k)}$ and that d has—by definition–the binary expansion $r = \chi(X_1)$ of width $(n-k)$.
   (b) The set S is just the subset $X_1 \subseteq V'$, and hence we get directly that $\chi(S) = \chi(X_1) = r$.
2. We get from Claim 2 that the polynomial $\mathcal{Q}_{(\ell-1)} = \mathcal{Q}_1 = \mathcal{P}_1$ contains the monomial $m = m_1 = x^{\chi(X_1)}$.

Induction step: $2 \leq \ell \leq k$

1. Here $m = m_1 m_2 \cdots m_\ell$. Let $m' = m_1 m_2 \cdots m_{(\ell-1)}$.
   (a) Let d be the degree of monomial m, and let d' be the degree of monomial m'. Let $S' = \biguplus_{1 \leq i \leq (\ell-1)} X_i$. By the inductive hypothesis, d' has a binary expansion r' of width $(n-k)$, and $\chi(S') = r'$. Also, $m_\ell = x^{\chi(X_\ell)}$ by definition. Since $m = m' \cdot m_\ell$ we get that $m = x^{d'} \cdot x^{\chi(X_\ell)} = x^{r'} \cdot x^{\chi(X_\ell)} = x^{\chi(S')} \cdot x^{\chi(X_\ell)} = x^{\chi(S') + \chi(X_\ell)}$, where we got the second expression by rewriting d' in binary. Now by assumption the sets S' and X_ℓ are disjoint subsets of V', and hence we get—see Lemma 1—that $d = \chi(S') + \chi(X_\ell) \leq 2^{(n-k)}$, and hence that d has a binary representation r of width $(n-k)$.

Algorithm 1 Algorithm for computing the sets $\mathcal{T}_i$ as bit strings T_i .

```
1: function ComputeTs(G, V', k)
2:     Create bit strings T_1, T_2, ..., T_k of length 2^(n-k) each, with all bits set to zero.
3:     for X ⊆ V' do
4:         for 1 ≤ i ≤ k do
5:             if (X ∪ {v_i}) is an independent set in G then
6:                 inTi ← True
7:                 for 1 ≤ j ≤ k, j ≠ i do
8:                     if (X ∪ {v_i, v_j}) is an independent set in G then
9:                         inTi ← False
10:                if inTi == True then
11:                    T_i[χ(X)] ← 1
12:    return (T_1, T_2, ..., T_k)
```

(b) Here $S = S' \uplus X_\ell$, and so from the above argument and Lemma 1 we get that $r = \chi(S)$.

2. We reuse notation from part (a) above. Let i be the Hamming weight of $\chi(S')$. Note that $i = |S'|$. Let $j = |X_\ell|$; then j is the Hamming weight of $\chi(X_\ell)$. Since S' and X_ℓ are disjoint subsets of the set V', we get from Lemma 1 that $i + j \leq (n - k)$. Therefore i, j is among the pairs of integers over which we iterate during the computation of the polynomial $\mathcal{Q}_\ell$. Let us examine carefully that step in this computation where we consider the pair i, j. Recall that we compute the polynomials $Q_i, P_j, R'_{ij}, R''_{ij}$, and R_{ij} in this step.

From the inductive assumption we get that (i) the polynomial $\mathcal{Q}_{(\ell-1)}$ contains the monomial $m' = m_1 m_2 \cdots m_{(\ell-1)}$, and (ii) $\chi(S')$ is the binary representation of the degree of m', and hence that i is the Hamming weight of the degree of m'. From these we get that the polynomial $Q_i = \mathcal{H}_i(\mathcal{Q}_{(\ell-1)})$ contains the monomial m'. Further, we have shown that the polynomial $\mathcal{P}_\ell$ contains the monomial $m_\ell = x^{\chi(X_\ell)}$, and by definition, j is the Hamming weight of the degree $\chi(X_\ell)$ of m_ℓ. From this we get that the polynomial $P_j = \mathcal{H}_j(\mathcal{P}_\ell)$ contains the monomial m_ℓ. Hence the product $R'_{ij} = Q_i \times P_j$, and thence its representative polynomial R''_{ij} both contain the monomial $m'm_\ell$.

From the inductive assumption we get that $m' = x^{\chi(S')}$, and by definition we have that $m_\ell = x^{\chi(X_\ell)}$. Thus $m'm_\ell = x^{\chi(S')+\chi(X_\ell)}$. Now since S' and X_ℓ are *disjoint* subsets of the set V', we get from Lemma 1 that $\chi(S') + \chi(X_\ell)$ has Hamming weight *exactly* $i + j$. Hence the monomial $m = m'm_\ell$ survives in the Hamming projection $R_{ij} = \mathcal{H}_{(i+j)}(R''_{ij})$. Therefore m is present in the polynomial $\mathcal{Q}_\ell$. ◀

▶ **Corollary 11** (⋆). *Let $m = \prod_{1 \leq \ell \leq k} m_i$. Then the polynomial $\mathcal{Q}_k$ contains m as a monomial, and the Hamming weight of the degree d of m is exactly $(n - k)$.*

Since $\mathcal{Q}_k$ contains the monomial $\prod_{1 \leq \ell \leq k} m_i$ whose degree has Hamming weight $(n - k)$, our algorithm returns Yes on this input. ◀

▶ **Lemma 12** (⋆). *If the algorithm of subsection 5.1 returns* Yes *on input (G, D), then graph G has a b-coloring with k colors for which D is a dominator set.*

Algorithm 2 Algorithm for computing the polynomial $\mathcal{Q}_\ell$ as a bit string S_ℓ.

```
1:  function ComputeQs(S_(ℓ−1), T_ℓ)
                          ▷ S_(ℓ−1), T_ℓ represent the polynomials Q_(ℓ−1), P_ℓ; see the text.
2:      Create bit strings P_1, P_2, ..., P_(n−k) of length 2^(n−k) each, with all bits set to zero.
3:      Create bit strings Q_1, Q_2, ..., Q_(n−k) of length 2^(n−k) each, with all bits set to zero.
4:      Create bit string S_ℓ of length 2^(n−k), with all bits set to zero.
5:      for 1 ≤ i ≤ 2^(n−k) do        ▷ Compute all the projections P_i and Q_i in one pass.
6:          h ← the Hamming weight of i
7:          if T_ℓ[i] == 1 then        ▷ The polynomial P_ℓ contains the monomial x^i
8:              P_h[i] ← 1
9:          if Q_(ℓ−1)[i] == 1 then    ▷ The polynomial Q_ℓ contains the monomial x^i
10:             Q_h[i] ← 1
11:     for 0 ≤ i ≤ (n − k) do
12:         for 0 ≤ j ≤ (n − k − i) do
13:             R''_ij ← FFT-Multiply(Q_i, P_j)        ▷ See explanation in text.
14:             for 1 ≤ p ≤ 2^(n−k) do
15:                 if the Hamming weight of p is i + j then
16:                     if R''_ij[p] == 1 then
17:                         S_ℓ[p] ← 1
18:     return S_ℓ
```

5.3 Running Time Analysis

▶ **Lemma 13.** *The algorithm of subsection 5.1 runs in $\mathcal{O}(2^{(n-k)}n^4 \log n)$ time where n is the number of vertices of the input graph G, and k is the size of the dominator set D.*

Proof. We assume that we are given the adjacency matrix of graph G as input. If required, we relabel the vertices of graph G as $V(G) = \{v_1, v_2, \ldots, v_n\}$ in such a way that the k vertices of the set D appear *last* in this list. In other words, such that $V' = V(G) \setminus D = \{v_1, v_2, \ldots, v_{(n-k)}\}$. This can be done in $\mathcal{O}(n)$ time.

We compute the sets $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$ as *bit strings* $T_1, T_2, \ldots, T_k$, respectively, of length 2^{n-k} each, with the following semantics: the j^{th} bit $T_i[j]$ of bit string T_i is 1 if and only if the subset $X \subseteq V'$ whose characteristic vector $\chi(X)$ with respect to V' satisfies the condition $val(X) = j$ is in the set $\mathcal{T}_i$. We use Algorithm 1 to compute these bit strings.

Creating the empty bit strings (line 2) takes $\mathcal{O}(2^{(n-k)})$ time. The innermost **for** loop (starting at line 7) has $\mathcal{O}(k)$ iterations, each of which takes $\mathcal{O}(n^2)$ time: this is the time we need for a brute-force check for independence of the set $X \cup \{v_i, v_j\}$ using the adjacency matrix of G. The **for** loop at line 7 therefore takes $\mathcal{O}(kn^2)$ time. Assuming that we can access any one bit of a $2^{(n-k)}$-sized bit string in time $\mathcal{O}(\log(2^{(n-k)})) = \mathcal{O}(n)$, line 11 takes $\mathcal{O}(n)$ time. The independence test of line 5 takes, as above,$\mathcal{O}(n^2)$ time. Thus the contents of the **for** loop at line 4 take $\mathcal{O}(n^2) + \mathcal{O}(kn^2) + +\mathcal{O}(n) = \mathcal{O}(kn^2)$ time. From this we get that the loop at line 3 takes $\mathcal{O}(2^{(n-k)} \cdot k \cdot kn^2) = \mathcal{O}(k^2n^2 2^{(n-k)})$ time. The algorithm for creating the bit strings $T_1, T_2, \ldots, T_k$ therefore takes $\mathcal{O}(k^2n^2 2^{(n-k)})$ time.

Observe now that each bit-string T_i serves *also* as a representation of the polynomial $\mathcal{P}_i$. This follows directly from the way we defined the polynomials $\mathcal{P}_i$ from the *sets* $\mathcal{T}_i$. Specifically: monomial x^d is present in polynomial $\mathcal{P}_i$ if and only if the bit $T_i[d]$ is set to 1. Hence we do not need to explicitly compute the polynomials $\mathcal{P}_i$; we just use the bit-strings T_i instead.

Recall that $\mathcal{Q}_1 = \mathcal{P}_1$. For each $2 \leq \ell \leq k$ we compute the polynomial $\mathcal{Q}_\ell$, again as a bit string S_ℓ of length $2^{(n-k)}$, as in Algorithm 2. We first set $S_1 = T_1$. For each $2 \leq \ell \leq k$, in increasing order of ℓ, we invoke the function ComputeQs with the arguments $S_{(\ell-1)}, T_\ell$ to obtain the bit string S_ℓ. The bit string S_k which we obtain at the end of this process is the representation of the polynomial $\mathcal{Q}_\ell$.

The function ComputeQs is mostly self-explanatory, except perhaps for lines 13 to 17. On line 13 we invoke the function **FFT-Multiply** which takes the bit string representations of two polynomials Q_i, P_j, computes the bit string representation of their product R'_{ij} using the Fast Fourier Transform, and returns the (bit string representation of the) representative polynomial R''_{ij} of R'_{ij} (see Lemma 2). In lines 14 to 17 we then add the Hamming projection $R_{ij} = \mathcal{H}_{(i+j)}(R''_{ij})$ to S_ℓ without explicitly computing R_{ij} as a separate bit string.

We now analyze the time taken by one invocation of ComputeQs. Lines 2 to 4 together take time $\mathcal{O}((n-k)2^{(n-k)})$. Lines 6 to 10 can each be executed in time $\mathcal{O}(n-k)$, and so the **for** loop at line 5 takes $\mathcal{O}((n-k)2^{(n-k)})$ time. Line 13 can be performed in $\mathcal{O}(2^{(n-k)}(n-k)\log(n-k))$ time (by Lemma 2), and the **for** loop at line 14 can be executed in $\mathcal{O}((n-k)2^{(n-k)})$ time as well. It follows that the **for** loop at line 11 takes $\mathcal{O}((n-k)^3 2^{(n-k)})$ time. In total, therefore, one invocation of ComputeQs takes $\mathcal{O}(2^{(n-k)}(n-k)^3 \log(n-k))$ time.

Since we invoke ComputeQs $(k-1)$ times, it follows that the complete algorithm runs in $\mathcal{O}(2^{(n-k)}(n-k)^3 k \log(n-k)) = \mathcal{O}(2^{(n-k)} n^4 \log n)$ time. ◀

6 Conclusion

In this paper we studied b-Chromatic Number in the realm of parameterized and exact algorithm and resolved the parameterized complexity of the problem. We would like to conclude with two concrete open problems.

- Is there an algorithm for b-Chromatic Number running in time $2^n n^{\mathcal{O}(1)}$?
- Does there exist an XP algorithm for b-Chromatic Number?
- Is the problem of finding a b-chromatic core of order k FPT when parameterized by k?

References

1 Kenneth Appel and Wolfgang Haken. The solution of the four-color-map problem. *Sci Am*, 237(4):108–121, October 1977.

2 Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM J. Computing*, 39(2):546–563, 2009.

3 Manfred Cochefert. *Algorithmes Exacts et Exponentiels pour les Problèmes NP-difficiles sur les Graphes et Hypergraphes*. PhD thesis, Université de Lorraine, December 2014.

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

5 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

6 Brice Effantin, Nicolas Gastineau, and Olivier Togni. A characterization of b-chromatic and partial grundy numbers by induced subgraphs. *CoRR*, abs/1505.07780, 2015.

7 Frantisek Galcík and Ján Katrenic. A note on approximating the b-chromatic number. *Discrete Applied Mathematics*, 161(7-8):1137–1140, 2013.

8 Frédéric Havet, Cláudia Linhares Sales, and Leonardo Sampaio. b-coloring of tight graphs. *Discrete Applied Mathematics*, 160(18):2709–2715, 2012.

9 Frédéric Havet and Leonardo Sampaio. On the grundy and b-chromatic numbers of a graph. *Algorithmica*, 65(4):885–899, 2013.

10 Robert W. Irving and David Manlove. The b-chromatic number of a graph. *Discrete Applied Mathematics*, 91(1-3):127–141, 1999.

11 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Plenum Press, New York, 1972.

12 Jan Kratochvíl, Zsolt Tuza, and Margit Voigt. On the b-chromatic number of graphs. In Ludek Kucera, editor, *Graph-Theoretic Concepts in Computer Science, 28th International Workshop, WG 2002, Cesky Krumlov, Czech Republic, June 13-15, 2002, Revised Papers*, volume 2573 of *Lecture Notes in Computer Science*, pages 310–320. Springer, 2002.

13 E. L. Lawler. A note on the complexity of the chromatic number problem. *Information Processing Lett.*, 5(3):66–67, 1976.

14 Arnold Schönhage and Volker Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7(3-4):281–292, 1971.

Fast Biclustering by Dual Parameterization

Pål Grønås Drange[1], Felix Reidl[2], Fernando Sánchez Villaamil[2], and Somnath Sikdar[2]

1 **Department of Informatics, University of Bergen, Norway**
pal.drange@ii.uib.no
2 **Theoretical Computer Science, RWTH Aachen University, Germany**
reidl,fernando.sanchez,sikdar@cs.rwth-aachen.de

Abstract

We study two clustering problems, Starforest Editing, the problem of adding and deleting edges to obtain a disjoint union of stars, and the generalization Bicluster Editing. We show that, in addition to being NP-hard, none of the problems can be solved in subexponential time unless the exponential time hypothesis fails.

Misra, Panolan, and Saurabh (MFCS 2013) argue that introducing a bound on the number of connected components in the solution should not make the problem easier: In particular, they argue that the subexponential time algorithm for editing to a fixed number of clusters (p-Cluster Editing) by Fomin et al. (J. Comput. Syst. Sci., 80(7) 2014) is an *exception* rather than the rule. Here, p is a secondary parameter, bounding the number of components in the solution.

However, upon bounding the number of stars or bicliques in the solution, we obtain algorithms which run in time $O(2^{3\sqrt{pk}} + n + m)$ for p-Starforest Editing and $O(2^{O(p\sqrt{k}\log(pk))} + n + m)$ for p-Bicluster Editing. We obtain a similar result for the more general case of t-Partite p-Cluster Editing. This is subexponential in k for a fixed number of clusters, since p is then considered a constant.

Our results even out the number of multivariate subexponential time algorithms and give reasons to believe that this area warrants further study.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases graph editing, subexponential algorithms, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.402

1 Introduction

Identifying clusters and biclusters has been a central motif in data mining research [22] and forms the cornerstone of algorithmic applications in, e.g., biology [25] and expression data analysis [7]. Cai [6] showed that clustering – among many other graph modification problems of similar flavor – is solvable in fixed-parameter tractable time. Parallel to these general results, some progress was made in the area of structurally sparse graphs: many problems are, when restricted to classes characterized by a finite set of forbidden minors, solvable in *subexponential parameterized time*, i.e. they admit algorithms with time complexity $2^{o(k)} \cdot \mathrm{poly}(n)$.

The complexity class of problems admitting such an algorithm is called SUBEPT and was defined by Flum and Grohe in the seminal textbook on parameterized complexity [14]. They simultaneously noticed that most natural problems did, in fact, *not* live in this complexity class: The classical NP-hardness reductions paired with the *exponential time hypothesis* of Impagliazzo, Paturi, and Zane [20] is enough to show that no $2^{o(k)} \cdot \mathrm{poly}(n)$ algorithm exists.

10th International Symposium on Parameterized and Exact Computation (IPEC 2015).
Editors: Thore Husfeldt and Iyad Kanj; pp. 402–413

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

In this context, Jianer Chen posed the following open problem in the field of parameterized algorithms [5]: Are there examples of natural problems on graphs, that do not have such a topological constraint, and also have subexponential parameterized running time? Alon, Lokshtanov, and Saurabh [1] partially answered this question in the positive by providing a subexponential time algorithm for Feedback Arc Set on tournament graphs. However, the aforementioned graph classes with topological constraints are sparse, and tournament graphs are extremely dense. Chen's question is therefore not fully answered – are there problems which are in SUBEPT on general graphs?

This is indeed the case. Fomin and Villanger [16] showed that Minimum Fill-In was solvable in time $2^{O(\sqrt{k}\log k)} + \text{poly}(n)$. Minimum Fill-In is the problem of completing a graph into a chordal graph by adding as few edges as possible. Following this, a line of research was established investigating whether more graph modification problems admit such algorithms. It proved to be a fruitful area; Since the result by Fomin and Villanger (ibid.), we now know that several graph modification problems towards classes such as split graphs [17], threshold graphs [10], trivially perfect graphs [11], (proper) interval graphs [3, 4], and more admit subexponential time algorithms.

While these classes are rather "simple", they certainly are much more complex than simple cluster or bicluster graphs. Therefore, the problems Cluster Editing and Cluster Deletion were a logical candidate for subexponential time algorithms. Surprisingly, we cannot expect that such algorithms exist. Komusiewicz and Uhlmann gave an elegant reduction proving that both parameterized and exact subexponential time algorithms were not achievable, unless ETH fails [21]. On the other hand, the problem p-Cluster Editing, where the number of components in the target class is fixed to be at most p – rather surprisingly – does indeed admit a subexponential parameterized time algorithm; This was shown by Fomin et al. [15], who designed an algorithm solving this problem in time $2^{O(\sqrt{pk})} \cdot \text{poly}(n)$.

Misra, Panolan, and Saurabh explicitly stated their surprise about this result: In their opinion, bounding the number of components in the target graph should in general *not* facilitate subexponential time algorithms [23]: *"We show that this sub-exponential time algorithm for the fixed number of cliques is rather an exception than a rule."*

We show that the related problem Bicluster Editing and its generalization t-Partite p-Cluster Editing as well as the special case Starforest Editing also belong to this exceptional class of problems where a bound on the number of target components greatly improves their algorithmic tractability. Since Bicluster Editing is an important tool in molecular biology and biological data analysis, and the necessary second parameter is not outlandish in these settings, we feel that this is a noteworthy insight. We complement these results with NP-completeness proofs for Bicluster Editing and t-Partite p-Cluster Editing on subcubic graphs and further show that, unless ETH fails, no parameterized or exact subexponential algorithm is possible without the secondary parameter. That a bound on the maximal degree does not contribute towards making these problems more tractable contrasts many other graph modification problems (like modifications towards split and threshold graphs [24]) which are polynomial-time solvable in this setting.

Previously, it was known Bicluster Editing in general is NP-complete [2], and Guo, Hüffner, Komusiewicz, and Zhang [18] studied the problem from a parameterized point of view, giving a linear problem kernel with $6k$ vertices, and an algorithm solving the problem in time $O(3.24^k + m)$.

Our contribution. In this paper, we study both the very general t-Partite p-Cluster Editing as well as editing to the aforementioned special cases. On the positive side, we show that

- p-Starforest Editing is solvable in time $O(2^{3\sqrt{pk}} + n + m)$, and
- both p-Bicluster Editing and the more general t-Partite p-Cluster Editing are solvable in time $2^{O(p\sqrt{k}\log(pk))} + O(n+m)$ facilitated by a kernel of size $O(ptk)$, where $t = 2$ in the case of p-Bicluster Editing.

In many cases, p is considered a constant, and in this case our kernel has size linear in k. We supplement these algorithms with hardness results; Specifically, we show that

- assuming ETH, Starforest Editing and Bicluster Editing cannot be solved in time $2^{o(k)} \cdot \text{poly}(n)$ and thus neither can t-Partite Cluster Editing, and finally,
- p-Starforest Editing is W[1]-hard if parameterized by p alone.

Organization of the paper. In Section 3 we give a subexponential time parameterized algorithm for the Starforest Editing problem when parameterized by the editing budget and the number of stars in the solution simultaneously. One ingredient for our subexponential algorithms is a polynomial kernel. A kernel for Bicluster Editing exists already [18] and we provide one for the t-partite case in Section 4. In Section 5 we show that p-Bicluster Editing is solvable in subexponential time in k; We give a $2^{O(p\sqrt{k}\log(pk))}+O(n+m)$ algorithm and generalize it to editing to t-partite p-cluster graphs. The parameter p is usually considered to be a fixed constant, hence the running time is truly subexponential, $2^{o(k)} + O(n+m)$ in the editing budget k. However, for a more fine-grained complexity analysis and for lower bounds, we treat p as a parameter.

In Section 6 we show that we cannot expect such an algorithm without an exponential dependency on p; The problem is not solvable in time $2^{o(k)}\,\text{poly}(n)$ unless ETH fails. Further, we show that Starforest Editing is W[1]-hard if parameterized by p alone, before we conclude in Section 7. Due to page limits, some proofs have been deferred to the full version, available online [12].

2 Preliminaries

We consider only finite simple graphs $G = (V, E)$ and we use n and m to denote the size of the vertex set and edge set, respectively. We denote by $N_G(v)$ the set of neighbors of v in G, and let $\deg_G(v) = |N_G(v)|$. We omit subscripts when the graph in question is clear from context. We refer to the monograph by Diestel [9] for graph terminology and notation not defined here. For information on parameterized complexity, we refer to the textbook by Flum and Grohe [14]. We consider an edge in $E(G)$ to be a set of size two, i.e., $e \in E(G)$ is of the form $\{u, v\} \subseteq V(G)$ with $u \neq v$. We denote by $[V(G)]^2$ the set of all size two subsets of G. When $F \subseteq [V(G)]^2$, we write $G \triangle F$ to denote $G' = (V, E \triangle F)$, where $\triangle$ is the *symmetric difference*, i.e., $E \triangle F = (E \setminus F) \cup (F \setminus E)$. When the graph is clear from context, we will refer to F simply as a set of edges rather than $F \subseteq [V(G)]^2$.

Let us fix the following terminology: A *star graph* is a tree of diameter at most two (a graph isomorphic to $K_{1,\ell}$ for some ℓ). The degree-one vertices are called *leaves* and the vertex of higher degree the *center*. A *starforest* is a forest whose connected components are stars or, equivalently, a graph that does not contain $\{K_3, P_4, C_4\}$ as induced subgraphs. A biclique is a complete bipartite graph $K_{a,b}$ for some $a, b \in \mathbb{N}$, and a *bicluster graph* is a disjoint union of bicliques. A t-partite clique graph is a graph whose vertex set can be partitioned into at most t independent sets, all pairwise fully connected, and a t-partite cluster graph is a

disjoint union of t-partite cliques. The problem of editing towards a starforest (resp. bicluster and t-partite cluster) is the algorithmic problem of adding and deleting as few edges as possible to convert a graph G to a starforest (resp. bicluster and t-partite cluster). We write $f(n) = \text{poly}(n)$ to mean $f(n) = n^{O(1)}$, i.e., that there exists a $c \in \mathbb{N}$ such that $f(n) = O(n^c)$.

Exponential time hypothesis. To show that there is no subexponential time algorithm for Starforest Editing we give a linear reduction from 3Sat, that is, a reduction which constructs an instance whose parameter is bounded linearly in the size of the input formula. The constructed instance will also have *size* bounded linearly in the size of the formula, and we use this to also rule out an exact subexponential algorithm of the form $2^{o(n+m)}$. Pipelining such a reduction with an assumed subexponential parameterized algorithm for the problem would give a subexponential algorithm for 3Sat, contradicting the complexity hypothesis of Impagliazzo, Paturi, and Zane [20]. Their Sparsification Lemma shows that, unless the exponential time hypothesis (ETH) fails, 3Sat cannot be solved in time $2^{o(n+m)}$, where n and m here refer to the number of variables and the number of clauses, respectively.

3 Editing to starforests in subexponential time

A first natural step in handling modification problems related to bicluster graphs is modification towards the subclass of bicluster graphs called starforest. Recall that a graph is a starforest if it is a bicluster where every biclique has one side of size exactly one, or equivalently, every connected component is a star.

Starforest Editing parameterized by k

Input:	A graph $G = (V, E)$ and a non-negative integer k.
Question:	Is there a set of at most k edges F such that $G \triangle F$ is a disjoint union of stars?

The problem where we only allow to *delete* edges is referred to as Starforest Deletion. These two problems can easily be observed to be equivalent; Adding an edge to a forbidden induced subgraph will create one of the other forbidden subgraphs, or simply put, it never makes sense to add an edge.

In Section 6 we show that this problem is NP-hard, and that it is not solvable in time $2^{o(k)}\,\text{poly}(n)$ unless the exponential time hypothesis fails.

Multivariate analysis. Since no subexponential algorithm is possible under ETH, we introduce a secondary parameter by p which bounds the number of connected components in a solution graph. This has previously been done with success in the Cluster Editing problem [15]. Hence, we define the following multivariate variant of the above problem.

p-Starforest Editing parameterized by p, k

Input:	A graph $G = (V, E)$ and a non-negative integer k.
Question:	Is there a set F of edges of size at most k such that $G \triangle F$ is a disjoint union of exactly p stars?

Observe that this problem is *not* the same as p-Starforest Deletion since we might need to merge stars to achieve the desired value p for the number of connected components. In Section 6 we show that the problem is W[1]-hard parameterized by p alone, and that we therefore need to parameterize on both p and k.

▶ **Lemma 1.** *Let (G, k) be input to p-Starforest Editing. If (G, k) is a yes-instance, there can be at most $p + 2k$ vertices with degree at least 2.*

The following bound will be key to obtain the subexponential running time.

▶ **Proposition 2** ([15])**.** *If a and b are non-negative integers, then $\binom{a+b}{a} \leq 2^{2\sqrt{ab}}$.*

▶ **Lemma 3.** *Given a graph G and a vertex set S, we can compute in linear time $O(n + m)$ an optimal editing set F such that $G \triangle F$ is a starforest, when restricted to have S as the set of centers in the solutions.*

We now describe an algorithm which solves p-Starforest Editing in time $O(2^{3\sqrt{pk}} + n + m)$.

The algorithm. Let (G, k) be an input instance for p-Starforest Editing. If the number of vertices of degree at least two is greater than $p+2k$, we say no in accordance with Lemma 1. Otherwise we split the graph into G_1 and G_2 as follows: Let $X \subseteq V(G)$ be the collection of vertices contained in connected components of size one or two, i.e., $G[X]$ is a collection of isolated vertices and edges. Let $G_1 = G[X]$ and $G_2 = G[V(G) \setminus X]$. Clearly, there are no edges going out of X in G. We will treat G_1, G_2 as (almost) independent subinstances by guessing the budgets $k_1 + k_2 = k$ and the number of components in their respective solutions $p_1 + p_2 = p$. The only time we cannot treat them as independent instances is when p_1 or p_2 is zero; Let p_i^* be the number of stars completely contained in G_i in an optimal solution. If both $p_i^* > 0$, then there always exist an optimal solution that does not add any edge between G_1 and G_2.

Solving (G_1, k_1) with p_1 components: Assume G_1 contains s isolated edges and t isolated vertices, with $p_1 > 0$. If $|V(G_1)| < p_1$, we immediately say no, since we need exactly p_1 connected components. Depending on the values of s and t, we execute the following operations as long as the budget k_1 is positive. If $s \leq p_1$ and $s + t \leq p_1$, we have too few stars, and we arbitrarily delete edges to increase the number of connected components to p_1.

If $s = 0$ we turn the isolated vertices arbitrarily into p_1 stars. Otherwise, fix an arbitrary endpoint c of an isolated edge. Assume that $s \leq p_1$: then we connect enough isolated vertices to c such that the number of stars is p_1. Finally, if $s > p_1$, we first dissolve $s - p_1$ edges and continue as in the previous case. It is easy to check that the above solutions are optimal.

Solving (G_2, k_2) with p_2 components: By Lemma 1, the number of vertices of degree at least two is bounded by $p_2 + 2k_2$. Every vertex of degree one in G_2 is adjacent to a vertex of larger degree, thus it never makes sense to choose it as a center (its neighbor will always be cheaper). Hence, it suffices to enumerate every set S_2 of p_2 vertices of degree larger than one and test in linear time, as per Lemma 3, whether a solution inside the budget k_2 is possible. Using Proposition 2 we can bound the running time by

$$\binom{p_2 + 2k_2}{p_2} \cdot pk + O(n + m) = O(2^{2\sqrt{2p_2k_2}} \cdot pk + n + m) = O(2^{3\sqrt{p_2k_2}} + n + m).$$

We are left with the cases where p_1 or p_2 are equal to zero: then the only possible solution is to remove *all* edges within G_1 or G_2, respectively, and connect all the resulting isolated vertices to an arbitrary center in the other instance. We either follow through with the operation, if within the respective budget, or deduce that the subinstance is not solvable. We conclude that the above algorithm will at some point guess the correct budgets for G_1 and G_2 and thus find a solution of size at most k. The theorem follows.

▶ **Theorem 4.** *p-Starforest Editing is solvable in time $O(2^{3\sqrt{pk}} + n + m)$.*

4 A polynomial kernel for t-partite p-cluster editing

We show a simple $O(ktp)$ kernel for the t-Partite p-Cluster Editing problem – which will be the foundation of the subsequent subexponential algorithms – with a single rule, Rule 1, which can be exhaustively applied in time $O(n+m)$. The problem at hand is the following generalization of p-Bicluster Editing:

t-Partite p-Cluster Editing parameterized by p, k

Input:	A graph $G = (V, E)$ and a non-negative integer k.
Question:	Is there a set $F \subseteq [V]^2$ of edges of size at most k such that $G \triangle F$ is a disjoint union of exactly p complete t-partite graphs?

For our rule, we say that a set $X \subseteq V(G)$ is a *non-isolate twin class* if for every v and v' in X, $N_G(v) = N_G(v') \neq \emptyset$. Note that this is by definition a *false twin class*, i.e., $vv' \notin E(G)$, or in other words, a non-isolate twin class is an independent set.

▶ **Rule 1.** *If there is a non-isolate twin class $X \subseteq V(G)$ of size at least $2k+2$, then delete all but $2k+1$ of them.*

▶ **Lemma 5.** *Rule 1 is sound and can be exhaustively applied in linear time.*

The following result is an immediate consequence of the above rule and its correctness.

▶ **Theorem 6.** *The problem t-Partite p-Cluster Editing admits a kernel where the number of vertices is bounded by $pt(2k+1)+2k = O(ptk)$.*

Proof. We now count the number of vertices we can have in a yes instance after the rule above has been applied. We claim that if G has more than $pt(2k+1)+2k$ vertices, it is a no instance. Let (G, k) be the reduced instance according to Rule 1 and let F be a solution of size at most k. At most $2k$ vertices can be touched by F, so the rest of the graph remains as it is, and is a disjoint union of at most p complete t-partite graphs, each of which has at most t non-isolate twin classes. It follows that in a yes instance, G has at most $pt(2k+1)+2k = O(ptk)$ vertices. ◀

5 Editing to bicluster graphs in subexponential time

In this section we lift the result of Section 3 by showing that the following problem is solvable in time $2^{O(p\sqrt{k}\log(pk))} + O(n+m)$. Observe that we lose the subexponential dependence on p, however, contrary to the result of Misra et al. [23], for fixed (or small, relative to k) p, this still is truly subexponential parameterized by k.

p-Bicluster Editing parameterized by p, k

Input:	A graph $G = (V, E)$ and a non-negative integer k.
Question:	Is there a set $F \subseteq [V]^2$ of edges of size at most k such that $G \triangle F$ is a disjoint union of p complete bipartite graphs?

We denote a biclique of G as $C = (A, B)$ and call the sets A, B the *sides* of C. Before describing the algorithm for the general problem, we show that the following simpler problem is solvable in linear time using a greedy algorithm:

Annotated Bicluster Editing

Input:	A bipartite graph $G = (A, B, E)$, a partition $\mathcal{A} = \{A_1, A_2, \ldots, A_p\}$ of A and a non-negative integer k.
Question:	Is there a set $F \subseteq [V]^2$ of edges of size at most k such that $G \triangle F$ is a disjoint union of p complete bipartite graphs with each one side in $\mathcal{A}$?

▶ **Lemma 7.** Annotated Bicluster Editing *is solvable in time* $O(n + m)$.

Proof. Let $G = (A, B, E)$, $\mathcal{A} = \{A_1, \ldots, A_p\}$, k be an instance of Annotated Bicluster Editing. Consider a vertex $v \in B$ and define $\text{cost}_i(v)$ to be the cost of placing v in B_i where $C_i = (A_i, B_i)$ is the ith biclique of the solution, i.e.,

$$\text{cost}_i(v) = |A_i| - 2\deg_{A_i}(v) + \deg(v),$$

where $\deg_{A_i}(v) = |N(v) \cap A_i|$. We prove the following claim which implies that we can greedily assign each vertex $v \in B$ to a biclique of minimum cost.

▶ **Claim 8.** *An optimal solution will always have* $v \in B$ *in a biclique* $C_i = (A_i, B_i)$ *which minimizes* $\text{cost}_i(v)$.

Suppose that $\text{cost}_i(v)$ is minimal but v is placed by a solution F in a biclique $C_j = (A_j, B_j)$ with $\text{cost}_j(v) > \text{cost}_i(v)$. Deleting from F all edges E_j between v and A_j and adding all edges E_i between v and A_i creates a new solution $F' = (F \setminus E_j) \cup E_i$. Since $\text{cost}_j(v) > \text{cost}_i(v)$, we have that $|F| > |F'|$ hence F is not optimal. This concludes the proof of the claim and the lemma. ◀

5.1 Subexponential time algorithm

We now show that the problem p-Bicluster Editing is solvable in subexponential time by using the kernel from Theorem 6, guessing the annotated sets and applying the polynomial time algorithm for the annotated version of the problem. The important ingredient will be *cheap* vertices, by which we mean vertices that are known to receive very few edits. Intuitively, a cheap vertex is a "pin" that in subexponential time reveals for us its neighborhood in the solution, and thus can be leveraged to uncover parts of said solution.

We adopt the following notation and vocabulary. For an instance (G, k) of p-Bicluster Editing, and a solution F, we call $H = G \triangle F$ the *target* graph. A vertex v is called *cheap* with respect to F if it receives at most $\sqrt{k}$ edits. Observe that any set X of size larger than $2\sqrt{k}$ has a cheap vertex. We call such a set *large* and all sets that contain at most $2\sqrt{k}$ vertices *small*. We will further classify the bicliques in the target graph into two different classes: A biclique is small if its vertex set is small and large otherwise.

The algorithm now works as follows. Given an input instance (G, k) of p-Bicluster Editing, we try all combinations of $p_s + p_\ell = p$, with the intended meaning that p_s is the number of small bicliques and p_ℓ is the number of large bicliques in the target graph.

Handling small bicliques. We enumerate a set of p_s sets $\mathcal{A}_s \subseteq 2^V$ with the property that they are pairwise disjoint, and each of size at most $2\sqrt{k}$. Furthermore, $G[\bigcup \mathcal{A}_s]$ contains at most k edges. Delete all edges in $\mathcal{A}_s$ and reduce the budget accordingly. These are going to be all the left sides in small bicliques. This enumeration takes time

$$(2\sqrt{k})^{p_s}\binom{n}{2\sqrt{k}}^{p_s} \leq (2\sqrt{k})^{p}\binom{pk+k^2}{2\sqrt{k}}^{p} = 2^{O(p\sqrt{k}\log(pk))}.$$

Handling large bicliques. The large bicliques have the following nice property. Since the vertex set of each such biclique is large, every biclique contains a cheap vertex. We guess a set $\mathcal{B}_\ell$ of size p_ℓ. For the biclique C_i, the vertex v_i of $\mathcal{B}_\ell$ will be a cheap vertex in B_i. Now, we enumerate all combinations of p_ℓ sets $\mathcal{N} = \langle N_1, N_2, \ldots, N_{p_\ell}\rangle$, each of size at most $2\sqrt{k}$ which will be the edited neighborhood of each cheap vertex, and we conclude that $A_i = N_H(v_i) = N_G(v_i) \triangle N_i$. The enumeration of this asymptotically takes time

$$\binom{n}{p_\ell} \cdot (2\sqrt{k})^{p_\ell} \binom{n}{2\sqrt{k}}^{p_\ell} \leq \binom{pk+k^2}{p} \cdot (2\sqrt{k})^{p} \binom{pk+k^2}{2\sqrt{k}}^{p} = 2^{O(p\sqrt{k}\log(pk))}.$$

Putting things together. With the above two steps, in time $2^{O(p\sqrt{k}\log(pk))}$ we obtained all the left sides $\mathcal{A}$, partitioned into $\mathcal{A}_s$ and $\mathcal{A}_\ell$. Using this information, we can in polynomial time compute whether the Annotated Bicluster Editing instance $(G, k, \mathcal{A})$ is a yes-instance. If so, we conclude yes, otherwise, we backtrack.

▶ **Theorem 9.** *p-Bicluster Editing is solvable in time $2^{O(p\sqrt{k}\log(pk))} + O(n+m)$.*

Proof. We now show that the algorithm described above correctly decides p-Bicluster Editing given an instance (G, k). Suppose that the algorithm above concludes that (G, k) is a yes instance. The only time it outputs yes, is when Annotated Bicluster Editing for a given set $\mathcal{A}$ and a given budget k' outputs yes. Since this budget is the leftover budget from making A an independent set, it is clear that any Annotated Bicluster Editing solution of size at most k' gives a yes instance for p-Bicluster Editing.

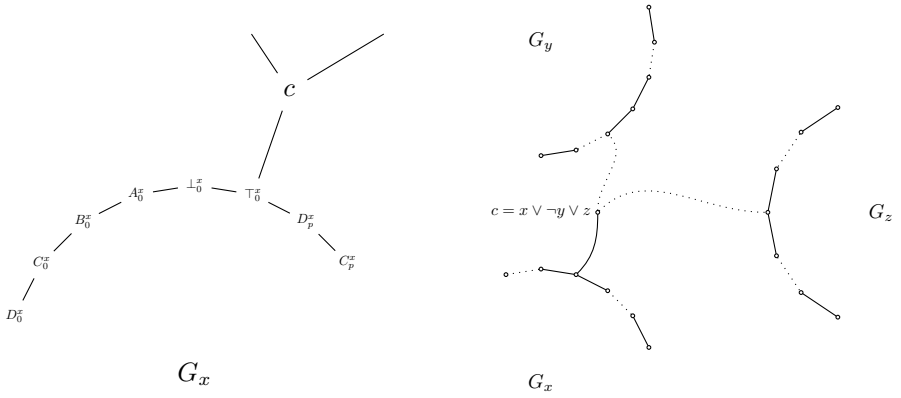
Suppose now for the other direction that (G, k) is a yes instance for p-Bicluster Editing and let F be a solution. Consider the left sides $A_1, \ldots, A_p$ of $G \triangle F$ with the restriction that the smaller of the two sides in C_i is named A_i. First we observe that during our subexponential time enumeration of sets, all the A_is that are of size at most $2\sqrt{k}$ will be enumerated in one of the branches where p_s is set to the number of small bicliques. Furthermore, if A_j is large, then both are large, and then, for each of the large bicliques, there is a branch where we selected exactly one cheap vertex for each of the largest sides. Given these cheap vertices, there is a branch where we guess exactly the edits affecting each of the cheap vertices, hence we can conclude that in some branch, we know the entire partition $\mathcal{A}$. From Lemma 7, we can conclude that the algorithm described above concludes correctly that we are dealing with a yes-instance. ◀

5.2 The t-partite case

We can in fact obtain similar (we treat t here as a constant so the results are up to some constant factors in the exponents) results for the more general case of t-Partite p-Cluster Editing. Again we need the polynomial kernel described in Theorem 6. The only difference now to the bicluster case is that we define a cluster to be small if *every side* is small. In this case, we can enumerate $\binom{n}{\sqrt{k}}^{tp}$ sets, which will form the small clusters.

In the other case a cluster $C = (A_1, A_2, \ldots, A_t)$ is divided into $A_1, A_2, \ldots, A_{t_s}$ small sides and $A_{t_s+1}, A_{t_s+2}, \ldots, A_t$ large sides. For this case, we guess *all* the small sides and for each of the large sides we guess a cheap vertex. Guessing the neighborhoods $N_{t_s+1}, N_{t_s+2}, \ldots, N_t$ for the cheap vertices $v_{t_s+1}, v_{t_s+2}, \ldots, v_t$ gives us complete information on C; To compute what A_j is, if $j > t_s$, we simply take the intersection $\bigcap_{t_s<i\leq t, i\neq j} N_i$ and remove $\bigcup_{i\leq t_s} A_i$. We arrive at the following lemma whose proof is directly analogous to that of Theorem 9.

▶ **Theorem 10.** *The problem t-Partite p-Cluster Editing is solvable in subexponential time $2^{O(p\sqrt{k}\log(pk))} + O(n+m)$.*



(a) Parts of a variable gadget x and its connection when occurring positively in c.

(b) Deletion when x is chosen to satisfy the clause. If we chose not to delete the edge connecting the clause vertex with G_y we would have gotten an induced P_4.

Figure 1 Reduction from 3Sat to starforest editing on subcubic graphs.

6 Lower bounds

We show that (a) Starforest Editing is NP-hard and that we cannot expect a subexponential algorithm unless the ETH fails; and (b) that p-Starforest Editing is W[1]-hard parameterized only by p.

6.1 Starforest editing

In the following we describe a linear reduction from 3Sat to Starforest Editing. Furthermore, the instance we reduce to has maximal degree three, thus not only showing that Starforest Editing is NP-hard on graphs of bounded degree, but also not solvable in subexponential time on subcubic graphs.

▶ **Theorem 11.** *The problem* Starforest Editing *is* NP*-complete and, assuming ETH, does not admit a subexponential parameterized algorithm when parameterized by the solution size k, i.e., it cannot be solved in time $2^{o(k)} \cdot \mathrm{poly}(n)$, nor in exact exponential time $2^{o(n+m)}$, even when restricted to subcubic graphs.*

To prove the theorem above we will reduce from 3Sat. But to obtain the result, it is crucial that in our reduction, both the parameter k, and the size of the instance G are bounded in linearly in n and m. Such results have been shown earlier, in particular by Komusiewicz and Uhlmann for Cluster Editing [21] and Drange and Pilipczuk for Trivially Perfect Editing [13]. Thus we resort to similar reductions as used there, however, the reductions here are tweaked to work for the problem at hand. We also achieve lower bounds for subcubic graphs. See Figures 1a and 1b for figures of the gadgets.

Variable gadget. Let φ be an input instance of 3Sat, and denote its variable set and clause set as $\mathcal{V}(\varphi)$ and $\mathcal{C}(\varphi)$, respectively. We construct for $x \in \mathcal{V}(\varphi)$ a graph $G_x \cong C_{6p_x}$ where p_x is the number of clauses in φ which x appears in. The vertices of G_x are labeled, consecutively, $\top_i^x, \perp_i^x, A_i^x, B_i^x, C_i^x, D_i^x$ for $i \in [0, p_x - 1]$.

There are exactly three ways of deleting G_x into a starforest using at most $k_x = 6p_x$ edges. Clearly a collection of P_3s is a starforest and is our target graph. We will define the $\top$-deletion for G_x as the deletion set $S^x_\top = \{C^x_i D^x_i, \bot^x_i A^x_i \mid i \leq p_x - 1\}$ and the $\bot$-deletion for G_x as the deletion set $S^x_\bot = \{A^x_i B^x_i, D^x_i \top^x_{i+1} \mid i \leq p_x - 1\}$, taking the $i+1$ in the index of $\top^x_{i+1}$ modulo p_x. In other words, in the gadget G_x, we are *keeping* the edges

- $D^x_{i-1}\top^x_i \bot^x_i, A^x_i B^x_i C^x_i$, when x is set to true, and
- $\top^x_i \bot^x_i A^x_i, B^x_i C^x_i D^x_i$, when x is set to false.

Observe that when x is set to true, we will have paths on three vertices, where $\top^x_i$ is the middle vertex, and if x is set to false, we will have paths on three vertices with $\bot^x_i$ being the middle vertex. Later, we will see that if x satisfies a clause c, the ith clause x appears in, then either $\top^x_i$ or $\bot^x_i$ will be the middle vertex of a claw, depending on whether x appears positively or negatively in c.

▶ **Observation 12.** *In an optimal edge edit of a cycle of length divisible by* 6*, no edge is added and exactly every third consecutive edge is deleted.*

Clause gadget. A clause gadget simply consists of one vertex, i.e., for a clause $c \in \mathcal{C}(\varphi)$, we construct the vertex v_c. This vertex will be connected to G_x, G_y and G_z, for x, y, z being its variables, in appropriate places, depending on whether or not the variable occurs negated in c. In fact, it will be connected to $\top^x_i$ if c is the ith clause x appears in, and x appears positively in c, and it is connected to $\bot^x_i$ if c is the ith clause x appears in, and x appears negatively in c.

Let $k_\varphi = 2|\mathcal{C}| + 2\sum_x p_x = 2|\mathcal{C}| + 3 \cdot 2|\mathcal{C}| = 8|\mathcal{C}|$ be the budget for a formula φ. We now observe that the budget is tight.

▶ **Lemma 13.** *The graph G_φ has no starforest editing set of size less than k_φ, and if the editing set has size k_φ it contains only deletions.*

We now continue to the main lemma, from which Theorem 11 follows.

▶ **Lemma 14.** *A* 3Sat *instance φ is satisfiable if and only if (G_φ, k_φ) is a yes instance for* Starforest Editing.

Observing that the maximum degree of G_φ is three – the clause vertices have exactly degree three, and the variable gadgets are cycles with some vertices connected to at most one clause vertex – this concludes the proof of Theorem 11. From the discussions above, the following result is an immediate consequence:

▶ **Corollary 15.** *The problem* Starforest Deletion *is* NP*-complete and not solvable in subexponential time under ETH, even on subcubic graphs.*

Before going into parameterized lower bounds of Starforest Editing, we show that the exact same reduction above simultaneously proves similar results for the bicluster case. We note that the NP-hardness was shown by Amit [2], but their reduction suffers a quadratic blowup and is therefore not suitable for showing subexponential lower bounds.

▶ **Corollary 16.** *The problems* Bicluster Editing *and* Bicluster Deletion *are* NP*-complete and not solvable in subexponential time under ETH, even on subcubic graphs.*

6.2 W[1]-hardness parameterized by p

In this section we show that the parameterization by k is necessary, even for the case of p-Starforest Editing. That is, we show that when we parameterize by p alone, the problem becomes W[1]-hard, and we can thus not expect any algorithms of the form $f(p)\cdot\text{poly}(n)$ for any function f solving p-Starforest Editing. We reduce from the problem Multicolored Regular Independent Set. An instance of this problem consists of a regular graph colored into p color classes, each color class inducing a complete graph, and we are asked to find an independent set of size p.

▶ **Proposition 17** ([8]). *The problem* Multicolored Regular Independent Set *is* W[1]*-complete.*

Since each color class is complete, any independent set will be of size at most p and any independent set of size p is maximum. The reduction is direct; In fact we have that given a budget $k = (n-p)(d-1)$, where d is the regularity degree, the following direct translation between the two problems holds:

▶ **Lemma 18.** *Let G be a d-regular graph on n vertices, $p \leq n$ and $k = (n-p)(d-1)$. Then (G,p) is a yes instance for* Multicolored Regular Independent Set *if and only if (G,k) is a yes instance for p-*Starforest Editing*.*

Combining Proposition 17 with Lemma 18 yields the following result:

▶ **Theorem 19.** *p-*Starforest Editing *is* W[1]*-hard when parameterized by p alone.*

7 Conclusion

We presented subexponential time algorithms for editing problems towards bicluster graphs, and more generally, t-partite cluster graphs when the number of connected components in the target graph is bounded. We supplemented these findings with lower bounds, showing that this dual parameterization is indeed necessary.

As an interesting open problem, we pose the question of whether t-Partite p-Cluster Editing can be solved in time $2^{O(\sqrt{pk})}\,\text{poly}(n)$, i.e., in subexponential time with respect to both parameters. It is known that Bicluster Editing admits a linear kernel, but when introducing the secondary parameter, we only obtain a kernel whose size is bounded by the product of both parameters; Recall that we got a $tp(2k+1)+2k$ kernel, which in the bicluster case is $p(4k+2)+2k$. Does Bicluster Editing admit a truly linear kernel, i.e., a kernel with $O(p+k)$ vertices?

Finally, in many practical applications of biclustering problems, the input can often be considered bipartite. The proof of the NP-completeness and subexponential algorithm lower bounds is highly non-bipartite, hence a natural question is whether it is possible to get similar lower bounds for the problem Bipartite Bicluster Editing, the problem where you are given a bipartite graph and asked to respect the bipartition.

Acknowledgments. We thank Daniel Lokshtanov for pointing us in the direction of the W[1]-completeness of the problem Regular Independent Set [8].

We would like to thank the anonymous reviewers for their feedback which greatly improved the quality of this paper.

Pål Grønås Drange has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 267959, *Rigorous Theory of Preprocessing.*

Fernando Sánchez Villaamil is supported by DFG Project RO 927/13-1, *Pragmatic Parameterized Algorithms.*

References

1 N. Alon, D. Lokshtanov, and S. Saurabh. Fast FAST. In *ICALP*, 2009.
2 N. Amit. The bicluster graph editing problem. Master's thesis, Tel Aviv University, 2004.
3 I. Bliznets, F. V. Fomin, M. Pilipczuk, and M. Pilipczuk. Subexponential parameterized algorithm for interval completion. *SODA*. ACM-SIAM, 2016. To appear.
4 I. Bliznets, F. V. Fomin, M. Pilipczuk, and M. Pilipczuk. A subexponential parameterized algorithm for proper interval completion. In *ESA*. Springer, 2014.
5 H. L. Bodlaender, L. Cai, J. Chen, M. R. Fellows, J. A. Telle, and D. Marx. Open problems in parameterized and exact computation, 2006. IWPEC.
6 L. Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Inf. Proc. Lett.*, 58(4):171–176, 1996.
7 Y. Cheng and G. M. Church. Biclustering of expression data. In *ISMB*. AAAI Press, 1999.
8 M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.
9 R. Diestel. *Graph theory (Graduate texts in mathematics)*. Springer, 2005.
10 P. G. Drange, M. Dregi, D. Lokshtanov, and B. D. Sullivan. On the threshold of intractability. In *ESA*. Springer, 2015.
11 P. G. Drange, F. V. Fomin, M. Pilipczuk, and Y. Villanger. Exploring subexponential parameterized complexity of completion problems. In *STACS*. Schloss Dagstuhl, 2014.
12 P. G. Drange, F. Reidl, F. Sánchez Villaamil, and S. Sikdar. Fast biclustering by dual parameterization. *CoRR*, abs/1507.08158, 2015.
13 P. G. Drange and M. Pilipczuk. A polynomial kernel for trivially perfect editing. In *ESA*. Springer, 2015.
14 J. Flum and M. Grohe. *Parameterized complexity theory*. Springer, 2006.
15 F. V. Fomin, S. Kratsch, M Pilipczuk, M. Pilipczuk, and Y. Villanger. Tight bounds for parameterized complexity of Cluster Editing with a small number of clusters. *J. Comput. Syst. Sci.*, 80(7):1430–1447, 2014.
16 F. V. Fomin and Y. Villanger. Subexponential parameterized algorithm for minimum fill-in. *SIAM J. Comput.*, 42(6):2197–2216, 2013.
17 E. Ghosh, S. Kolay, M. Kumar, P. Misra, F. Panolan, A. Rai, and M. S. Ramanujan. Faster parameterized algorithms for deletion to split graphs. *Algorithmica*, 71(4):989–1006, 2015.
18 J. Guo, F. Hüffner, C. Komusiewicz, and Y. Zhang. Improved algorithms for bicluster editing. In *TAMC*. Springer, 2008.
19 M. Habib and C. Paul. A survey of the algorithmic aspects of modular decomposition. *Computer Science Review*, 4(1):41–59, 2010.
20 R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.
21 C. Komusiewicz and J. Uhlmann. Cluster editing with locally bounded modifications. *Discrete Appl. Math.*, 160(15):2259–2270, 2012.
22 S. C. Madeira and A. L Oliveira. Biclustering algorithms for biological data analysis: a survey. *IEEE/ACM Trans. Comput. Biol. Bioinform*, 1(1):24–45, 2004.
23 N. Misra, F. Panolan, and S. Saurabh. Subexponential algorithm for d-cluster edge deletion: Exception or rule? In *MFCS*. Springer, 2013.
24 A. Natanzon, R. Shamir, and R. Sharan. Complexity classification of some edge modification problems. *Discrete Appl. Math.*, 113(1):109–128, 2001.
25 A. Tanay, R. Sharan, and R. Shamir. Biclustering algorithms: A survey. *Handbook of Comp. Mol. Biol.*, 9(1-20):122–124, 2005.