Parameterized Algorithms for Graph Modification Problems

by

Ashutosh Rai
MATH10201005006

The Institute of Mathematical Sciences, Chennai

A thesis submitted to the
Board of Studies in Mathematical Sciences
(Theoretical Computer Science)
In partial fulfillment of requirements
for the Degree of

DOCTOR OF PHILOSOPHY

of

HOMI BHABHA NATIONAL INSTITUTE

July, 2016

Homi Bhabha National Institute

Recommendations of the Viva Voce Board

As members of the Viva Voce Board, we certify that we have read the dissertation prepared by Ashutosh Rai entitled “Parameterized Algorithms for Graph Modification Problems” and recommend that it maybe accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

____ Date:

Chair - Venkatesh Raman

____ Date:

Guide/Convener - Saket Saurabh

____ Date:

Member 1 - Meena Mahajan

____ Date:

Member 2 - V. Arvind

____ Date:

Member 3 - N. R. Aravind

Final approval and acceptance of this dissertation is contingent upon the candidate’s submission of the final copies of the dissertation to HBNI.

I hereby certify that I have read this dissertation prepared under my direction and recommend that it may be accepted as fulfilling the dissertation requirement.

Date:

Place: Guide

STATEMENT BY AUTHOR

Ashutosh Rai

DECLARATION

I, hereby declare that the investigation presented in the thesis has been carried out by me. The work is original and has not been submitted earlier as a whole or in part for a degree / diploma at this or any other Institution / University.

Ashutosh Rai

To my parents and my sister

ACKNOWLEDGEMENTS

First and foremost, I want to express my heartfelt gratitude to Professor Saket Saurabh, my advisor, and Professor Venkatesh Raman, my co-advisor, for their guidance, support and time. I feel very fortunate to have got the opportunity to work under them, which always made me aspire to learn more. My special and sincere thanks goes to Professor Saket Saurabh for being available for all my questions anytime. He has guided me during my stay at IMSc in more ways than I can count. I am highly grateful for his belief in me and for pushing me to reach my potential.

Secondly, I am grateful to all the faculty at the Department of Theoretical Computer Science at IMSc, Professors V. Arvind, Kamal Lodaya, Meena Mahajan, R. Ramanujam, Vikram Sharma and C. R. Subramaniam. Whatever I know about Computer Science, I owe it to them.

I would like to thank Professors Gregory Gutin and Stefan Szeider for their invitations to collaborate, which resulted in a very enriching experience for me. I would also like to thank my institute, The Institute of Mathematical Sciences for providing me with funding which enabled me to travel and attend various conferences and workshops, where I could learn more.

I acknowledge and thank my co-authors without whom this work might not have been possible. I am much obliged to M. S. Ramanujan, Daniel Lokshtanov, Neeldhara Misra, Geevarghere Philip and Manu Basawaraju, from whom I have learnt a lot during our discussions at various points. I would also like to thank Mark Jones, Rober Crowston, Gabriele Mucciacia and Neha Lodha who took every effort to make my stay enjoyable in London and Vienna respectively.

I am thankful to Fahad Panolan, Pranabendu Misra, Sudeshna Kolay, Syed Meesum, Gaurav Rattan, Nitin Saurabh, Prafullakumar Tale and Roohani Sharma for creating an environment in the institute which was always conducive for discussions.

I consider myself lucky to have supporting friends and would like to thank all of them, specially Nitesh, Anup, Ramanathan, Nikita, Madhu and Smriti, who have been with me through all the highs and lows and ensured that the times were not dull. Lastly and most importantly, I owe a lot to my family, who took care of all other things, so that I could concentrate on my work here.

Contents

Abstract

This thesis investigates some graph modification problems from Parameterized Complexity point of view. A typical graph modification problem, for a fixed graph class Π, asks us to modify the input graph using small number of operations such that the resulting graph belongs to Π. Typical operations studied in the field are vertex and edge deletions. There are two choices of parameters for graph modification problems, one is the *size of the graph* we are looking for, the other is the *editing distance*.

The first part of the thesis deals with the former kind of parameterization and has results concerning many choices of the graph class Π. The first result establishes no polynomial kernelization under standard complexity theory assumptions for finding induced hereditary subgraphs for many choices of Π, including cographs, chordal, interval, split, perfect, and cluster graphs. In the other result, the class Π is the set of q-colorable graphs. We give efficient FPT algorithms for finding induced q-colorable subgraphs on graphs where either all maximal independent sets can be enumerated in polynomial time or where the maximum independent set can be found in polynomial time.

The second part of the thesis deals with the more conventional parameter choice, which is the edit distance, more commonly called the solution size. We first give efficient FPT algorithms for Split Vertex Deletion and Split Edge Deletion when parameterized by the solution size. We also give polynomial kernels for both the problems. Then we consider the problem of deleting *both* vertices and edges to get a forest, which is a generalization of classic Feedback Vertex Set problem, and show that it is FPT.

Then we propose another parameterization for graph editing problems where after deleting a small number of vertices, we want *every connected component* of the resulting graph to be close to a well understood graph class Π, where the measure of closeness is the minimum number of edges to be deleted from that connected component to reach the graph class. We argue how this parameterization is more powerful than the standard parameterizations for graph editing problems, and show this version to be FPT for two choices of Π, forests and bipartite graphs. While showing the latter, we also develop an algorithm for a generalization of the classic Min-Cut problem, called Mixed Cut, where we are allowed to delete both vertices and edges to disconnect the given terminals. We also show that Mixed Cut is NP–complete.

List of Publications in this Thesis

Journals

1. Faster Parameterized Algorithms for Deletion to Split Graphs. Esha Ghosh, Sudeshna Kolay, Mrinal Kumar, Pranabendu Misra, Fahad Panolan, Ashutosh Rai, and M. S. Ramanujan. *Algorithmica*, 2015, Volume 71(4), pages 989-1006.

2. Kernel Lower Bounds using Co-nondeterminism: Finding Induced Hereditary Subgraphs. Stefan Kratsch, Marcin Pilipczuk, Ashutosh Rai, and Venkatesh Raman. *TOCT*, 2014, Volume 7(1), pages 4:1-4:18.

Conferences

1. Faster Parameterized Algorithms for Deletion to Split Graphs. Esha Ghosh, Sudeshna Kolay, Mrinal Kumar, Pranabendu Misra, Fahad Panolan, Ashutosh Rai, and M. S. Ramanujan. In *13th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2012, Helsinki, Finland, July 4-6, 2012, Proceedings*, pages 107-118, 2012.

2. Kernel Lower Bounds using Co-nondeterminism: Finding Induced Hereditary Subgraphs. Stefan Kratsch, Marcin Pilipczuk, Ashutosh Rai, and Venkatesh Raman. In *13th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2012, Helsinki, Finland, July 4-6, 2012, Proceedings*, pages 364-375, 2012.

3. Parameterized Algorithms for Max Colorable Induced Subgraph problem on Perfect Graphs. Neeldhara Misra, Fahad Panolan, Ashutosh Rai, Venkatesh Raman, and

Saket Saurabh. In *Graph-Theoretic Concepts in Computer Science - 39th International Workshop, WG 2013, Lübeck, Germany, June 19-21, 2013, Revised Papers*, pages 370-381, 2013.

4. Bivariate Complexity Analysis of Almost Forest Deletion. Ashutosh Rai and Saket Saurabh. In *Computing and Combinatorics - 21st International Conference, COCOON 2015, Beijing, China, August 4-6, 2015, Proceedings*, pages 133-144, 2015.

5. Generalized Pseudoforest Deletion: Algorithms and Uniform Kernel. Geevarghese Philip, Ashutosh Rai, and Saket Saurabh. In *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part II, volume 9235 of Lecture Notes in Computer Science*, pages 517-528. Springer, 2015.

6. A Parameterized Algorithm for Mixed-Cut. M. S. Ramanujan, Ashutosh Rai, and Saket Saurabh. In *LATIN 2016: Theoretical Informatics - 12th Latin American Symposium, Ensenada, Mexico, April 11-15, 2016, Proceedings*, pages 672-685, 2016.

7. Strong Graph Deletion: Bipartite Graphs. M. S. Ramanujan and Ashutosh Rai. In *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016, December 13-15, 2016, Chennai, India*, pages 21:1-21:14, 2016.

Ashutosh Rai

List of Figures

Part I

Introduction

Chapter 1

Introduction

Graph modification problems have been a central area of study in the field of algorithms. Many of the classical problems, including Vertex Cover, Feedback Vertex Set, Clustering and many others, can be looked as instances of a generic problem, where we are asked to modify the input graph G using small number of operations such that the resulting graph G' satisfies certain properties (or, in other words, belongs to a graph class Π). Most of these problems are NP–complete due to a general result of Lewis and Yannakakis [LY80], and hence have been approached from different paradigms designed to deal with NP-completeness, such as approximation algorithms and parameterized algorithms. In rest of the section, we define the abstract graph editing problem and mention the relevant results, which will stablish the foundation for describing the main results in the thesis.

In the field of parameterized complexity, graph editing problems feature prominently. The Vertex Cover problem, where we want to delete minimum number of vertices to get an independent set, has been one of the most researched problems in this field. Another graph editing problem which contributed to the advancement of the field in recent times is the Feedback Vertex Set problem, where we want to delete minimum number of vertices to get an acyclic graph. Both of these problems are known to be FPT and admit polynomial kernels. In this thesis, we concern ourselves with graph problems where we are only allowed to *delete* vertices or edges (or both) to get a graph with certain properties. In parameterized complexity, these deletion problems can be broadly classified into two types depending on the choice of the parameter. Now we describe the two classifications along with some relevant results.

1. **Size of the graph G'.** In this view, the parameter is the size of the graph we are looking for. It can also be looked at as the problem of *finding induced subgraphs.* We call the problem Π-Induced Subgraph and define it formally as follows.

Π-Induced Subgraph	**Parameter(s):** k
Input: A graph G and an integer k.	
Question: Does there exist an induced subgraph of G on k vertices that belongs to Π?	

Khot and Raman [KR02] proved the following result about Π-Induced Subgraph.

Theorem 1.1 ([KR02]). *If a nontrivial and hereditary graph class* Π *does not contain all independent sets and cliques,* Π-Induced Subgraph *becomes W[1]-hard, and is* FPT *otherwise.*

The kernelization complexity of Π-Induced Subgraph had remained largely unexplored until recently. After the development of theory of lower bounds on kernelization, Kratsch [Kra12] showed that Π-Induced Subgraph does not admit polynomial kernels unless NP ⊆ coNP/poly, where the class Π is the class of all independent sets and cliques.

2. The editing distance. In this view, the parameter is the editing distance. This can be the number of vertices, or edges, or both. We define the problem formally as follows.

$\mathcal{F}$-Deletion	**Parameter(s):** k_1, k_2
Input: An undirected graph G and non-negative integers k_1 and k_2.	
Question: Does there exist $S_1 \subseteq V(G)$, and $S_2 \subseteq E(G)$ such that $\lvert S_1\rvert \leq k_1, \lvert S_2\rvert \leq k_2$ and $G - S_1 - S_2$ is in $\mathcal{F}$?	

Here $G - S_1 - S_2$ is the graph obtained by deleting S_1 and S_2 from G. We call the problem $\mathcal{F}$-Vertex Deletion and $\mathcal{F}$-Edge Deletion in the special cases where the values of k_2 and k_1 respectively are zero. In a general result about $\mathcal{F}$-Vertex Deletion, Cai [Cai96] showed the following.

Theorem 1.2 ([Cai96]). *$\mathcal{F}$-*Vertex Deletion *is* FPT *for all hereditary graph classes $\mathcal{F}$*

which can be characterized by a finite forbidden set of graphs as induced subgraphs.

The theorem shows that $\mathcal{F}$-Vertex Deletion is FPT for a wide class of choices of $\mathcal{F}$, including Independent Set, Triangle-free graphs, Split Graphs, Cographs etc. The FPT algorithm essentially finds a forbidden induced subgraph and branches on it. We can get a similar result for edge-deletion version of the problems by branching on the edges instead. Also, all these problems admit polynomial kernelization by reduction to the Hitting Set problem, where the size of the kernel depends on the size of the largest forbidden induced subgraph.

There are many other graph families, which are hereditary but forbid an infinite set of graphs as induced subgraphs. A few examples of such families are Bipartite graphs, Chordal graphs, Interval graphs, Perfect graphs etc. For such families, finding out whether $\mathcal{F}$-Vertex Deletion is FPT requires a lot more work and it has been an active area of research in the field of parameterized complexity.

In the next couple of sections, we describe the contents of the thesis and the relevance of those results in the general framework of graph editing problems.

1.1 Thesis Outline

In this thesis, we explore the parameterized complexity of various problems from both point of views as described in the previous section. In addition to that, we give a new way of parameterizing graph editing problems and explore the complexity of some problems under that particular view. In this section, we briefly mention the results in the thesis. In the next section, we discuss the results in more detail and give an organization of the thesis.

We show that Π-Induced Subgraph does not admit polynomial kernelization unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$ for two choices of graph class Π, which cover many well known graph classes including Cographs, Chordal graphs, Perfect graphs, Interval graphs, Cluster graphs etc.

Then we give two algorithms for Π-Induced Subgraph, where the class Π is the set of q-colorable graphs. These algorithms run in FPT time for graph classes where (*i*) the number of maximal independent sets is polynomial in the size of the graph, or (*ii*) the maximum independent set can be found in polynomial time. We also show that the problem does not admit polynomial kernel on perfect graphs unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

We look at the $\mathcal{F}$-Vertex Deletion and $\mathcal{F}$-Edge Deletion problems for the case when the $\mathcal{F}$ is the set of Split graphs, and give fast parameterized algorithms for both the problems. We also give better polynomial kernels for both problems.

Then we look at the general $\mathcal{F}$-Deletion problem for where the graph class $\mathcal{F}$ is the set of forests. We show that the problem is FPT and give polynomial kernel for the problem. We also give subexponential parameterized algorithm for the problem on planar graphs using bidimensionality.

We also introduce the notion of Strong $\mathcal{F}$-Deletion problem, where we want to delete at most k vertices such that *every connected component* of the resulting graph is at most ℓ edges away from being in $\mathcal{F}$. We show that Strong $\mathcal{F}$-Deletion deletion problem is FPT for the choices of $\mathcal{F}$ being forests and bipartite graphs.

1.2 Results and thesis organization

In this section, we describe the results in the chapters of the thesis in more detail. The technical content of the thesis is organized in the chapters which are divided into two parts as described in the subsections below.

1.2.1 Finding Induced Subgraphs

Kernel lower bounds for finding induced hereditary subgraphs

We first give an overview of the existing lower bound techniques for kernelization which will be used in the subsequent chapters.

Given the characterization in Theorem 1.1, we explore the existence of polynomial kernelizations for Π-Induced Subgraph, when Π is restricted to contain all independent sets and cliques. Regarding Π-Induced Subgraph, we show that for most natural graph classes Π, including cographs, chordal, interval, split, perfect, and cluster graphs, there is no polynomial kernelization unless NP ⊆ coNP/poly. This is shown by lower bounds for two choices for Π which are established by a co-nondeterministic cross-composition and a parameterized reduction from Ramsey respectively.

As one of the tools for our compositions we establish a nice trick for allowing easier source problems, which is of independent interest for other lower bounds. We show that for two general classes of problems, modeled after monotone and anti-monotone optimization problems, it suffices to start from *improvement versions*, where each instance comes with a guaranteed solution which is only off by one from the target value.

The results in this chapter are reported in [KPRR14].

Finding max q-colorable induced subgraph

We define the p-Max Colorable Induced Subgraph (p-mcis) problem formally as follows.

p-Max Colorable Induced Subgraph (p-mcis) **Parameter(s):** ℓ
Input: An undirected graph $G = (V, E)$ and positive integers q and ℓ.
Question: Does there exist $Z \subseteq V$, $|Z| \geq \ell$, such that $G[Z]$ is q-colorable?

Theorem 1.1 implies that p-mcis is W[1]-hard parameterized by the solution size on general graphs. Observe that Independent Set is essentially p-mcis with $q = 1$. There has been also some study of parameterized complexity of Independent Set on special graph classes [DLMR10, RS08]. Yannakakis and Gavril [YG87] showed that p-mcis is NP–complete on split graphs and Addario-Berry et al. [ABKK^{+}10] showed that the problem is NP–complete on perfect graphs for every fixed $q \geq 2$.

Our main contributions are two randomized FPT algorithms for p-mcis and a complementary lower bound, which establishes the non-existence of a polynomial kernel under standard complexity-theoretic assumptions.

Our first algorithm runs in time $(2e)^{\ell}(n + \#\alpha(G))^{O(1)}$ where $\#\alpha(G)$ is the number of maximal independent sets of the input graph and the second algorithm runs in time $O(6.75^{\ell+o(\ell)}n^{O(1)})$ on graph classes where the maximum independent set of an induced subgraph can be found in polynomial time. The first algorithm is efficient when the input graph contains only polynomially many maximal independent sets; for example on split graphs and co-chordal graphs. The second algorithm is efficient for a larger class of graphs, for example perfect graphs, because it only relies on an efficient procedure for finding a maximum independent set (although this comes at the cost of slightly larger base of the exponent).

We also describe derandomization procedures. While the derandomization technique for the first algorithm is standard, to derandomize the second algorithm we used the idea of (n, p, q)-separating families, introduced in [FLS14]. Further, we show that unless NP $\subseteq$ coNP/poly, the problem does not admit polynomial kernel even on split graphs. Also, on perfect graphs, we show that the problem does not admit a polynomial kernel even for fixed $q \geq 2$, unless NP $\subseteq$ coNP/poly.

The results in this chapter are reported in [MPR+13].

1.2.2 Graph Editing Problems

Deletion to Split graphs

We look at Split Vertex Deletion and Split Edge Deletion problems where we are allowed to delete at most k vertices and edges respectively to get to a split graph. The formal problems are defined as follows.

Split Vertex Deletion (SVD) **Parameter(s):** k

Input: Graph $G = (V, E)$, integer k

Question: Does there exist a set of vertices of size at most k whose deletion from G results in a split graph?

Split Edge Deletion (SED) **Parameter(s):** k

Input: Graph $G = (V, E)$, integer k

Question: Does there exist a set of edges of size at most k whose deletion from G results in a split graph?

As the forbidden set (as induced subgraphs) for split graphs is finite ($\{C_4, C_5, 2K_2\}$), these problems become fixed-parameter tractable due to a Theorem 1.2, when parameterized by k. One can also observe from the finite forbidden (induced) subgraph characterization of split graphs, a fairly straightforward branching algorithm for both SVD and SED with $\mathcal{O}^*(5^k)$ running time[1]. In [LNR+14], the authors obtained an $\mathcal{O}^*(2.32^k)$ algorithm for SVD by reducing the problem to the Above Guarantee Vertex Cover problem and using the fixed-parameter algorithm for it. We improve this bound to $\mathcal{O}^*(2^k)$ by the combination of a bound on the number of split partitions of a split graph, and the well known technique of iterative compression. We also obtain an $\mathcal{O}(k^3)$ vertex kernel for the problem. Note that, this kernel is smaller than the kernel with $\mathcal{O}(k^4)$ vertices, which can be obtained by an approach similar to d-Hitting Set [AK10]. We also prove that under certain complexity theoretic assumptions, we cannot obtain a subexponential algorithm for this problem.

For SED, we design a subexponential algorithm running in time $\mathcal{O}^*(2^{\mathcal{O}(\sqrt{k}\log k)})$ by combining the color and conquer approach [ALS09], with the bound on the number of partitions of a split graph. This was probably the second problem (see [FV12]) having a subexponential algorithm on general graphs which does not use bidimensionality theory. We also revisit the kernelization algorithm for this problem given by Guo [Guo07], and by using only a subset of the rules presented there, we prove a bound of $\mathcal{O}(k^2)$ vertices improving on Guo's bound of $\mathcal{O}(k^4)$. Furthermore, the Split Completion problem of adding at most k edges to a given graph to make it split, is equivalent to deleting at most k edges from the complement of the graph to make it split. Hence, the bound on the kernel and the subexponential algorithm which we prove for SED also holds for Split Completion. Later the algorithms for SVD and SED were improved to $\mathcal{O}^*(1.2738^k k^{\mathcal{O}(\log k)})$ and $\mathcal{O}^*(2^{\mathcal{O}(\sqrt{k})})$ respectively [CP13, CFK+15].

[1]Here the $\mathcal{O}^*$ notation hides the polynomial factors

The results of this chapter are reported in [GKK+15].

Almost Forest Deletion

We look at the $\mathcal{F}$-Deletion problem where the family $\mathcal{F}$ is set of forests. We call a graph an *ℓ-forest* if we can delete at most ℓ edges from it to get to a forest. The problem is formally defined as follows.

Almost Forest Deletion **Parameter(s):** k, ℓ

Input: A graph G, integers ℓ and k.

Question: Does there exist $X \subseteq V(G)$ such that $|X| \leq k$ and $G - X$ is an ℓ-forest?

We show that Almost Forest Deletion can be solved in $\mathcal{O}^*(5.0024^{(k+\ell)})$ time. We arrive at the result using the iterative compression technique which was introduced in [RSV04] and a non-trivial measure which helps us in getting the desired running time. Then we explore the kernelization complexity of the problem, and show that Almost Forest Deletion admits a polynomial kernel with $\mathcal{O}(k\ell(k+\ell))$ edges. For arriving at the result, we first make use of Expansion Lemma and Gallai's Theorem for reducing the maximum degree of the graph, and then we bound the size of the graph. It is easy to see that for Yes instances of Almost Forest Deletion, the treewidth of the input graph is bounded by $k+\ell$. Since we have an algorithm with running time of the form $\mathcal{O}^*(c^{(k+\ell)})$ on general graphs, the question of finding an algorithm with running time $\mathcal{O}^*(c^{(\mathbf{tw})})$ becomes interesting, where **tw** is the treewidth of the input graph. We answer this question affirmatively by giving a $\mathcal{O}^*(17^{\mathbf{tw}})$ algorithm for graphs with a tree decomposition of width **tw**. This algorithm, along with the notion of bidimensionality, gives rise to a subexponential algorithm for Almost Forest Deletion on planar graphs.

The results of this chapter appear in [RS15].

ℓ-Pseudoforest Deletion

We study the problem of deleting at most k vertices such that in the resulting graph, *every connected component* is at most ℓ edges away from being acyclic. It can also be looked at as deleting at most k vertices to get to the graph class $\mathcal{F}$, which is set of graphs where every connected component is at most ℓ edges away from being a forest. We define the problem formally as follows.

ℓ-PSEUDOFOREST DELETION	**Parameter(s):** k, ℓ
Input: A graph G, integers ℓ and k.	
Question: Does there exist $X \subseteq V(G)$ such that $G - X$ is an ℓ-pseudoforest?	

Here ℓ-pseudoforest represents the graph class where we can delete at most ℓ edges from each connected component to get to a forest. In our work, we first apply the techniques used in [FLMS12] to get an FPT algorithm for ℓ-PSEUDOFOREST DELETION. To that end, we have to show that ℓ-PSEUDOFOREST DELETION has a protrusion replacer, which we do by showing that the property of being an ℓ-pseudoforest is strongly monotone and minor-closed. We arrive at the running time of $\mathcal{O}^*(c_\ell^k)$ for ℓ-PSEUDOFOREST DELETION where c_ℓ is a function of ℓ alone. If we try to apply the machinery of [FLMS12] to get a kernelization algorithm for ℓ-PSEUDOFOREST DELETION, it only gives a kernel of size k^c where the constant c depends on ℓ. We use the similarity of this problem with FEEDBACK VERTEX set and applied Gallai's theorem and Expansion Lemma to decrease the maximum degree of the graph. This, when combined with techniques used in [FLMS12], gives us a kernel of size ck^2, where the constant c depends on ℓ. These kind of kernels are more desired as it gives $\mathcal{O}(k^2)$ kernel for every fixed ℓ, while the non-uniform kernelization does give a polynomial kernel for every fixed ℓ, but the exponent's dependency on ℓ makes the size of the kernel grow very quickly when compared to uniform-kernelization case.

We also look at a special case of ℓ-PSEUDOFOREST DELETION, namely PSEUDOFOREST DELETION, where we ask whether we can delete at most k vertices to get to a *pseudoforest*. A pseudoforest is special case of ℓ-pseudoforest for $\ell = 1$, *i.e.*, in a pseudoforest, each connected component is just one edge away from being a tree. In other words, it is a class of graphs where every connected component has at most one cycle. We applied the well known technique of iterative compression along with a non-trivial measure and an interesting base case to arrive at an $\mathcal{O}^*(7.5618^k)$ algorithm for this problem. We also gave an explicit kernel with $\mathcal{O}(k^2)$ vertices for the problem.

The results of this chapter are reported in [PRS15].

Pseudobipartite Deletion

Let $\mathcal{F}$ be a polynomial-time recognizable family of graphs; that is, given a graph G, in polynomial time we can decide whether G belongs to $\mathcal{F}$. For a fixed integer ℓ, let $\mathcal{F} + \ell e$ denote the class of graphs that can be obtained by adding (or deleting) at most ℓ edges to a graph in $\mathcal{F}$. We introduce the notion of STRONG $\mathcal{F}$-DELETION as follows.

STRONG $\mathcal{F}$-DELETION **Parameter(s):** k_1, k_2

Input: An undirected graph G and non-negative integers k_1 and k_2.

Question: Does there exist $S_1 \subseteq V(G)$ such that $|S_1| \leq k_1$ and every *connected component* of $G - S_1$ belongs to $\mathcal{F} + k_2 e$?

We first argue that this is an interesting and stronger parameterization for graph deletion problems as compared to $\mathcal{F}$-DELETION. In this chapter, we look at an instantiation of STRONG $\mathcal{F}$-DELETION where the class $\mathcal{F}$ is the class of bipartite graphs. Formally, we look at the following problem.

PSEUDOBIPARTITE DELETION **Parameter(s):** k, ℓ

Input: A graph G, integers k and ℓ and a set $U \subseteq V(G)$.

Question: Does there exist $X \subseteq (V(G) \setminus T)$ such that $|X| \leq k$, $X \cap U = \emptyset$ and $G - X$ is ℓ-pseudobipartite?

Here, an ℓ-pseudobipartite graph is defined analogous to ℓ-pseudoforests, that is, a graph is ℓ-pseudobipartite if we can delete at most ℓ edges from each of its connected components such that every connected component of the resulting graph is bipartite. In our work, we show that PSEUDOBIPARTITE DELETION is FPT. The first big obstacle that needed to be overcome was the fact that the input graph can have a minimum odd cycle transversal of unbounded size. The first part of our algorithm is devoted to overcoming this obstacle.

We utilise the recursive understanding technique introduced by Chitnis et al. [CCH$^+$12] to first find a small separator in the graph which separates the graph into two parts, each of sufficiently large size and then recursively solve a 'border' version of the same problem on one of the two sides. The border version of the problem is a generalization which also incorporates a special bounded set of vertices, called terminals. During the course of our algorithm, we attempt to solve the border problem on various subgraphs of the input graph. The objective in the border problem is to find a bounded set of vertices contained within a particular subgraph such that any vertex in this subgraph *not* in the computed set is not required in *any* solution for the given instance irrespective of the vertices chosen outside this subgraph.

Given the output of the border problem, the standard approach is to either delete the remaining vertices or simply 'bypass' these vertices. In our case such a reduction was not possible and no simple reduction seemed likely. However, we show that if we blow up the size of the computed set by a function of the parameter then we can use a 'parity preserving' bypassing to get rid of the remaining vertices.

This leaves us with the base case of the recursion, that is when we are unable to find a separator of the required kind. At this stage, we argued that if the instance is a yes-instance then it must in fact have an *odd cycle transversal (oct)* whose size is bounded by a function of the parameter. Interestingly, even this seemingly significant structural information regarding the input was not enough to guarantee an algorithm. Instead, we computed an approximate oct solution and designed a branching rule that has as its base case, the case when the approximate oct is not separated by the solution. Here, we rephrased this problem as a cut problem, namely MMCU, which we show to be FPT in the next chapter.

The results in this chapter are reported in [RR16].

Mixed Multiway Cut-Uncut

Given a graph, a typical *cut problem* asks for finding a set of vertices or edges such that their removal from the graph makes the graph satisfy some separation property. The most fundamental version of cut problems is Minimum Cut, where given a graph and two vertices, called *terminals*, we are asked to find the minimum sized subset of vertices (or edges) of the graph such that deleting them separates the terminals. While Minimum Cut is polynomial time solvable; even a slight generalization becomes NP-hard. Two of the most studied generalizations of Minimum Cut problem which are NP-hard are Multiway Cut and Multicut. In the Multiway Cut problem, we are given a set of terminals, and we are asked to delete minimum number of vertices (or edges) to separate the terminals from each other. This problem is known to be NP-hard even when the number of terminals is at least three. In the Multicut problem, given pairs of terminals, we are asked to delete minimum number of vertices (or edges) so that it separates all the given terminal pairs. The Multicut problem is known to be NP-hard when the number of pairs of terminals is at least three. The mixed version of the problem, where we are allowed to delete both edges and vertices, namely Mixed Cut, is also NP-hard.

In the Mixed Cut problem we are given an undirected graph G, vertices s and t, positive integers k and ℓ and the objective is to test whether there exist a k sized vertex set $S \subseteq V(G)$ and an ℓ sized edge set $F \subseteq E(G)$ such that deletion of S and F from G disconnects s from t. In this work, we study the problem called Mixed Multiway Cut-Uncut (MMCU), which is generalization of Mixed Cut, where given a graph G, $T \subseteq V(G)$, and an equivalence relation $\mathcal{R}$ on T and integers k and ℓ, we are asked whether there exist $X \subseteq (V(G) \setminus T)$ and $F \subseteq E(G)$ such that $|X| \leq k$, $|F| \leq \ell$ and for all $u, v \in T$, u and v belong to the same connected component of $G - (X, F)$ if and only if $(u, v) \in \mathcal{R}$, where $G - (X, F)$ is the graph that we obtain by deleting X and F from G. It is easy to see that this problem generalizes both, the Multiway Cut and Mixed Cut problems. The formal definition is given as below.

Mixed Multiway Cut-Uncut (MMCU) **Parameter(s):** k, ℓ

Input: A multigraph G, a set of terminals $T \subseteq V(G)$, an equivalence relation $\mathcal{R}$ on the set T and integers k and ℓ.

Question: Does there exist $X \subseteq (V(G) \setminus T)$ and $F \subseteq E(G)$ such that $|X| \leq k$, $|F| \leq \ell$ and for all $u, v \in T$, u and v belong to the same connected component of $G - (X, F)$ if and only if $(u, v) \in \mathcal{R}$?

Even though the vertex and edge versions of Minimum Cut problem are polynomial time solvable, we show that allowing deletion of both, the vertices and the edges, makes the Mixed Cut problem NP-hard. To show that, we use a simple reduction from the Bipartite

PARTIAL VERTEX COVER problem which was recently shown to be NP-hard [AS14, JV12]. Then we show that MMCU is FPT when parameterized by $k + l$ using the technique of recursive understanding introduced by Grohe et al. [GKMW11] (also see [CCH+12]).

The results of this chapter are reported in [RRS16].

Chapter 2

Preliminaries

2.1 Sets and numbers

We use $\mathbb{N}$ to denote the set of natural numbers. For any $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, 2, \ldots, n\}$. For a set U and $k \in \mathbb{N}$, we use 2^U and $\binom{U}{k}$ to denote the set of all subsets of U and set of all subsets of size k of U, respectively. A *partition* of a positive integer x is a multiset $P = \{x_1, x_2, \ldots, x_t\}$ such that $t \leq x$, $\sum_{i \in [t]} x_i = x$ and $x_i \in \mathbb{Z}^+$. The positive integer $x_i \in P$ is called *part* of the partition.

2.2 Growth of Functions

We mainly use the big-Oh ($\mathcal{O}$) notation (see [CSRL01]) and the big-Oh-star ($\mathcal{O}^*$) notation introduced in [Woe01]. Let $f : \mathbb{N} \to \mathbb{N}$ and $g : \mathbb{N} \to \mathbb{N}$ be two functions. We say that $f(n) = \mathcal{O}(g(n))$ if there exist constants c and n_0 such that for all $n \geq n_0$, $f(n) \leq cg(n)$. The notation $\mathcal{O}^*$ is essentially the $\mathcal{O}$ notation which hides the polynomial factors and used only for running times of exponential time algorithms. We use $\mathcal{O}^*(f(n))$ to denote $\mathcal{O}(f(n)n^c)$ where c is some constant. In this thesis, we use the $\mathcal{O}^*$ notation to hide factors which are polynomial in the input size in order to focus on the function of the parameter.

2.3 Graphs

A *graph* G is a pair (V, E) where V is a set of elements called *vertices*, and E is a collection of pairs of vertices called the *edges*. The two vertices of an edge are called the *endpoints* of the edge and, the edge is said to be *incident* on the two vertices. For a graph G we use $V(G)$ and $E(G)$ to denote the vertex set and the edge sets respectively. When the graph G is clear from the context, we use n and m to represent $|V(G)|$ and $|E(G)|$ respectively. For a graph $G = (V, E)$, and a set $A \subseteq E$ of edges, we denote by $V(A)$ the set of endpoints of the edges in A. A *simple graph* is a graph where every edge is distinct and further the two endpoints of any edge are distinct vertices. A *multigraph* is a graph where we allow multiple copies of the same edge in the edge set (*parallel edges*) and further, we allow an edge to have the same vertex as both it's endpoints (*loops*). A *subdivision* of an edge $e = (u, v)$ of G yields a new digraph, G', containing one new vertex w, and with an edge set replacing e by two new edges, (u, w) and (w, v). That is, $V(G') = V(G) \cup \{w\}$ and $E(G') = (E(G) \setminus \{(u, v)\}) \cup \{(u, w), (w, v)\}$. Unless specified otherwise, the graphs in this thesis are simple.

For $X \subseteq V(G)$, $G[X]$ denotes the *induced subgraph* on X of G, which has vertex set X, and the edge contains all edges in G whose both endpoints lie in X. Similarly, the *subgraph of G induced by an edge set $A \subseteq E$* is defined as the subgraph of G with edge set A and vertex set $V(A)$ and is denoted by $G[A]$. Similarly, the *subgraph of G induced by an edge set $A \subseteq E$* is defined as the subgraph of G with edge set A and vertex set $V(A)$ and is denoted by $G[A]$. All vertices adjacent to a vertex v are called neighbors of v and the set of all such vertices is called the neighborhood of v. Similarly, a non-adjacent vertex of v is called a non-neighbor and the set of all non-neighbors of v is called the non-neighborhood of v. The neighborhood of v is denoted by $N(v)$. We say that v is *global* to a set Z if v is adjacent to all vertices of Z and we say that v is non-adjacent (or non-neighbor) to a set Z if v is not adjacent to any vertex of Z. For two sets X and Y, we say that X is *global* to Y if every vertex in X is global to Y and that X is non-adjacent to Y if every vertex in X is non-adjacent to Y. Given a graph or digraph G and a subset X of $V(G)$ (or $E(G)$), by $G - X$ we denote the graph obtained by deleting X from $V(G)$ (or from $E(G)$). When X

contains just a single vertex v (or a single edge e), we often write $G - v$ (and $G - e$) to denote $G \setminus \{v\}$ (and $G \setminus \{e\}$).

The *union of graphs* G_1 and G_2 is the graph with vertex set $V(G_1) \cup V(G_2)$ and edge set $E(G_1) \cup E(G_2)$ A graph G_1 is said to be a subgraph of another graph G_2 is $V(G_1) \subseteq V(G_2)$ and $E(G_1) \subseteq E(G_2)$. A *walk* W in a graph G consists of a sequence of vertices and edges $\{v_0, e_1, v_1, e_2, \ldots, e_\ell, v_\ell\} \subseteq E(G)$, such that the edge e_i is incident on v_{i-1} and v_i. We say that a walk W visits the vertices and the edges in W. The vertices v_0 and v_ℓ are called the start and the end vertex, respectively, of the walk. All other vertices visited by W are called internal vertices. A walk is a closed walk if it's start vertex and end vertex are the same. A path P is a walk which visits any vertex at most once, i.e. there are at most two arcs in P which are incident on any vertex visited by W. A cycle is a closed walk which visits all the internal vertices exactly once and the start/end vertex exactly twice. Observe that any edge or vertex occurs at most once in a path P which induces an ordering of these edges, and we say that P visits these edges in that order. Similarly, for the collection of vertices which are present in P, P induces ordering of these vertices and we say that P visits them in that order. Let P be a path which visits a vertex u and then visits a vertex v. We write $P[u, v]$ to denote the sub-path of P which starts from u and ends at v. For two path P and Q such that the end vertex of P is same as the start vertex of Q, we write $P + Q$ to denote the walk from the start vertex of P to the end vertex of Q. A *cycle* is a graph with vertex set $V(P) = \{v_1, v_2, \ldots, v_\ell\}$ and edge set $E(P) = \{(v_i, v_{i+1}) \mid 1 \leq i \leq \ell - 1\} \cup \{(v_\ell, v_1)\}$, i.e. it is a path plus an edge from the last to the first vertex. A *acyclic graph*, as the name implies, contains no cycles. A connected acyclic graph is called a *tree*. An acyclic graph is a union of trees, and it is called a *forest*. A *clique* or a complete graph is a simple graph where every pair of vertices form an edge. A clique on n vertices is denoted by K_n. An independent set is a graph with an empty edge set.

A graph G is called *perfect*, if for all $U \subseteq V(G)$, $w(G[U]) = \chi(G[U])$. A graph G is called *chordal* if every simple cycle of with more than three vertices has an edge connecting two non-consecutive vertices on the cycle. The complements of chordal graphs are called *co-chordal* graphs. All chordal graphs and co-chordal graphs are perfect graphs. A *split* *graph* is a graph whose vertex set can be partitioned into two subsets I and Q such that I is an independent set and Q is a clique. Split graphs are closed under complementation.

2.4 Parameterized Complexity

In the parameterized complexity setting, an instance comes with an integer parameter k — formally, a parameterized problem $\mathcal{Q}$ is a subset of $\Sigma^* \times \mathbb{N}$ for some finite alphabet Σ. We say that the problem is *fixed-parameter tractable (*FPT*)* if there exists an algorithm solving any instance (x, k) in time $f(k)\text{poly}(|x|)$ for some (usually exponential or worse) computable function f. Fixed-parameter tractability can equivalently be defined via the notion of *kernelization*: a kernelization algorithm for a problem $\mathcal{Q}$ takes any instance (x, k) and in time polynomial in $|x| + k$ produces an equivalent instance (x', k') (i.e., $(x, k) \in \mathcal{Q}$ iff $(x', k') \in \mathcal{Q}$) such that $|x'| + k' \leq g(k)$ for some computable function g. The function g gives an upper bound on the *size of the kernel*, and if it is polynomial, we say that $\mathcal{Q}$ admits a polynomial kernel.

Theorem 2.1 (Folklore)**.** *A parameterized problem is* FPT *if and only if it admits a kernelization algorithm.*

In view of Theorem 2.1, every parameterized problem has a kernel, but the kernels obtained in this way turn out to be exponential. Hence, for problems already known to be FPT the question of having a *polynomial* kernelization becomes relevant, and has led to a lot of research in the field. Over time, many problems are shown to admit polynomial kernels, and using some recent techniques, some are ruled out to have one. In the later chapters, we will explore the lower bound techniques for kernelization in detail and will also see some examples of polynomial kernelizations.

If we do not demand the output of a kernelization to be an instance of the same problem but allow any (unparameterized) target language (accordingly only the output *size* is bounded by $g(k)$) then we obtain the notion of a (polynomial) *compression.* The significance of the latter is that almost all lower bound techniques in fact give lower bounds for this more general notion and, furthermore, lower bounds for compressions transfer more cleanly via reductions (and immediately also rule out kernels of the same size).

2.5 Treewidth

Let G be a graph. A *tree-decomposition* of a graph G is a pair $(\mathbb{T}, \mathcal{X} = \{X_t\}_{t \in V(\mathbb{T})})$ such that

- $\cup_{t \in V(\mathbb{T})} X_t = V(G)$,
- for every edge $xy \in E(G)$ there is a $t \in V(\mathbb{T})$ such that $\{x, y\} \subseteq X_t$, and
- for every vertex $v \in V(G)$ the subgraph of $\mathbb{T}$ induced by the set $\{t \mid v \in X_t\}$ is connected.

The *width* of a tree decomposition is $\mathsf{max}_{t \in V(\mathbb{T})} |X_t| - 1$ and the *treewidth* of G is the minimum width over all tree decompositions of G and is denoted by $\mathbf{tw}(G)$.

A tree decomposition $(\mathbb{T}, \mathcal{X})$ is called a *nice tree decomposition* if $\mathbb{T}$ is a tree rooted at some node r where $X_r = \emptyset$, each node of $\mathbb{T}$ has at most two children, and each node is of one of the following kinds:

1. **Introduce node**: a node t that has only one child t' where $X_t \supset X_{t'}$ and $|X_t| = |X_{t'}| + 1$.
2. **Forget node**: a node t that has only one child t' where $X_t \subset X_{t'}$ and $|X_t| = |X_{t'}| - 1$.
3. **Join node**: a node t with two children t_1 and t_2 such that $X_t = X_{t_1} = X_{t_2}$.
4. **Base node**: a node t that is a leaf of $\mathbb{T}$, is different than the root, and $X_t = \emptyset$.

Notice that, according to the above definition, the root r of $\mathbb{T}$ is either a forget node or a join node. It is well known that any tree decomposition of G can be transformed into a nice tree decomposition maintaining the same width in linear time [Klo94]. We use G_t to denote the graph induced by the vertex set $\cup_{t'} X_{t'}$, where t' ranges over all descendants of t, including t. By $E(X_t)$ we denote the edges present in $G[X_t]$. For clarity of presentation we use the term nodes to refer to the vertices of the tree $\mathbb{T}$.

Part II

Algorithms and Kernelization Lower Bounds and for Induced Subgraph Problems

Chapter 3

Lower Bounds for Kernelization: The basics

3.1 Introduction

In chapter 2, we have seen the definition of kernelization and how the notion of kernelization is related to that of Fixed Parameter Tractability. The theory of kernelization has developed recently to establish some strong connections with classical complexity, which has led to a framework for ruling out polynomial kernels under some complexity theoretic assumptions. Almost all kernelization lower bound results and tools are, directly or indirectly, based on a framework due to Bodlaender et al. [BDFH09] and Fortnow and Santhanam [FS11]. On a high level the framework centers around the fact that NP-hard problems cannot have both a polynomial kernelization and a so-called OR-composition algorithm (see Section 3.2 for basic definitions) unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$ (known to cause a collapse of the polynomial hierarchy).

In this chapter, we will broadly overview the results and the techniques which will be used in the next few chapters to arrive at lower bounds for kernelization.

3.2 The Framework

The two main ways of showing kernelization and compression lower bounds are to either give some variant of composition or to transfer a lower bound from another problem to the target problem by an appropriate reduction. Next we provide the essential definitions for both strategies, starting with OR-cross-compositions.

Definition 3.1 (polynomial equivalence relation [BJK14]). *Let Σ be a finite alphabet. An equivalence relation $\mathcal{R} \subseteq \Sigma^* \times \Sigma^*$ is called a* polynomial equivalence relation *if the following two conditions hold.*

1. *There is an algorithm that given two strings $x, y \in \Sigma^*$ takes time polynomial in $|x|+|y|$ and correctly answers whether they are equivalent under $\mathcal{R}$, i.e., whether $(x, y) \in \mathcal{R}$.*
2. *For any finite set $S \subseteq \Sigma^*$ the number of $\mathcal{R}$-equivalence classes in S is bounded polynomially in the size of the largest string in S.*

The idea behind polynomial equivalence relations is that it suffices to give OR-cross-compositions that work for any single equivalence class. We are now ready to give the definition of OR-cross composition.

Definition 3.2 (OR-cross-composition [BJK14]). *Let $L \subseteq \Sigma^*$ be a language and let $\mathcal{Q} \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem. We say that L* has a OR-cross-composition of cost $f(t)$ into *$\mathcal{Q}$ (where $f(t)$ is some computable function) if there is a polynomial equivalence relation $\mathcal{R} \subseteq \Sigma^* \times \Sigma^*$ and a deterministic algorithm C that, on input of t instances $x_1, \ldots, x_t \in \Sigma^*$ of L that are equivalent under $\mathcal{R}$, takes time polynomial in $\sum_i |x_i|$ and outputs on each computation path an instance $(y, k) \in \Sigma^* \times \mathbb{N}$ such that the following three conditions hold.*

"PB" *The parameter value k is bounded by $f(t)(\mathsf{max}_i|x_i|)^c$ on each computation path, for some constant c independent of t.*

"OR:Yes" *If* at least one *instance x_i is* YES *for L then $C(x_1, \ldots, x_t)$ returns an instance (y, k) that is* YES *for $\mathcal{Q}$.*

"OR:No" *If* all *instances x_i are* No *for L then $C(x_1, \ldots, x_t)$ returns an instance (y, k) that is* No *for $\mathcal{Q}$.*

We then call C a OR-cross-composition from L into $\mathcal{Q}$ with respect to $\mathcal{R}$.

We define the notion of the *unparameterized version* of a parameterized problem Π. The mapping of parameterized problems to unparameterized problems is done by mapping (x, k) to the string $x\#1^k$, where $\# \in \Sigma$ denotes the blank letter and 1 is an arbitrary letter in Σ. In this way, the unparameterized version of a parameterized problem Π is the language $\tilde{\Pi} = \{x\#1^k | (x, k) \in \Pi\}$. If we use the $\tilde{\Pi}$ as the source problem in OR-cross-composition to get to an instance of Π, that is if we have a OR-cross-composition from $\tilde{\Pi}$ to Π then we simply call it OR-*composition* instead of OR-*cross-composition.*

Now, we introduce co-nondeterministic OR-cross-compositions; note that, apart from the co-nondeterministic behavior, these follow the same style of definition as standard OR-cross-composition, which is defined above.

Definition 3.3 (co-nondeterministic OR-cross-composition [Kra12])**.** *Let $L \subseteq \Sigma^*$ be a language and let $\mathcal{Q} \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem. We say that L* has a co-nondeterministic OR-cross-composition of cost $f(t)$ into $\mathcal{Q}$ *(where $f(t)$ is some computable function) if there is a polynomial equivalence relation $\mathcal{R} \subseteq \Sigma^* \times \Sigma^*$ and a nondeterministic algorithm C that, on input of t instances $x_1, \ldots, x_t \in \Sigma^*$ of L that are equivalent under $\mathcal{R}$, takes time polynomial in $\sum_i |x_i|$ and outputs on each computation path an instance $(y, k) \in \Sigma^* \times \mathbb{N}$ such that the following three conditions hold.*

"PB" *The parameter value k is bounded by $f(t)(\mathsf{max}_i|x_i|)^c$ on each computation path, for some constant c independent of t.*

"OR:Yes" *If* at least one *instance x_i is* Yes *for L then* each *computation path of $C(x_1, \ldots, x_t)$ returns an instance (y, k) that is* Yes *for $\mathcal{Q}$.*

"OR:No" *If* all *instances x_i are* No *for L then* at least one *computation path of $C(x_1, \ldots, x_t)$ returns an instance (y, k) that is* No *for $\mathcal{Q}$.*

We then call C a co-nondeterministic OR-cross-composition from L (or *coNP-OR-cross-composition*) into $\mathcal{Q}$ with respect to $\mathcal{R}$*.*

The following theorem establishes the desired lower bound consequences of having a coNP-OR-cross-composition (or an OR-cross-composition) of low cost into some target problem $\mathcal{Q}$, i.e., ruling out polynomial kernels and compressions for $\mathcal{Q}$ unless $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$. The key fact is that a coNP-OR-cross-composition (or an OR-cross-composition) of some language L into a parameterized problem $\mathcal{Q}$ combined with a (possibly co-nondeterministic) polynomial kernelization for $\mathcal{Q}$ would give a *co-nondeterministic* weak OR-distillation for L. As observed by Chen and Müller the proof technique of Fortnow and Santhanam [FS11] still applies for co-nondeterministic OR-distillations and implies $L \in \mathsf{coNP/poly}$ (cf. [HN10]).

Theorem 3.4 ([Kra12])**.** *Let $L \subseteq \Sigma^*$ be a language, let $\mathcal{Q} \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem, and let $\mathcal{R} \subseteq \Sigma^* \times \Sigma^*$ be a polynomial equivalence relation. If L has a co-NP-OR-cross-composition with respect to $\mathcal{R}$ with cost $f(t) = t^{o(1)}$ (for t instances) and $\mathcal{Q}$ has a co-nondeterministic polynomial compression, then $L \in \mathsf{coNP/poly}$. If L is NP-hard then* $\mathsf{NP} \subseteq \mathsf{coNP/poly}$ *and the polynomial hierarchy collapses to its third level.*

The second lower bound strategy, instead of a direct proof via compositions, is to provide a polynomial parameter transformation from a problem with an established lower bound.

Definition 3.5 ([BTY11])**.** *Let $\mathcal{P}, \mathcal{Q} \subseteq \Sigma^* \times \mathbb{N}$ be parameterized problems. We say that a polynomially computable function $f\colon \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$ is a* polynomial parameter transformation *(PPT) from $\mathcal{P}$ to $\mathcal{Q}$ if for all $(x, k) \in \Sigma^* \times \mathbb{N}$ the following holds: $(x, k) \in \mathcal{P}$ if and only if $(x', k') = f(x, k) \in \mathcal{Q}$ and $k' \leq k^{\mathcal{O}(1)}$.*

Coupled with a Karp reduction (i.e., standard polynomial many-one reduction) from $\mathcal{Q}$ to $\mathcal{P}$, a polynomial parameter transformation from $\mathcal{P}$ to $\mathcal{Q}$ transfers any polynomial kernel result from $\mathcal{Q}$ to $\mathcal{P}$ [BTY11]. For our purposes of transferring lower bounds it suffices that a PPT from $\mathcal{P}$ to $\mathcal{Q}$ alone already transfers polynomial compressions (combined with the fact that almost all lower bounds rule out polynomial kernels *and* polynomial compressions). Thus, if $\mathcal{P}$ has no polynomial compression, e.g., under the assumption that $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$, then neither has $\mathcal{Q}$.

Theorem 3.6. *Let $\mathcal{P}$ and $\mathcal{Q}$ be parameterized problems and assume that there is a polynomial parameter transformation from $\mathcal{P}$ to $\mathcal{Q}$. If $\mathcal{Q}$ admits a polynomial kernel or compression, then $\mathcal{P}$ admits a polynomial compression. (Hence if $\mathcal{P}$ admits no polynomial compression under some assumption then neither does $\mathcal{Q}$.)*

Chapter 4

Kernel lower bounds for finding induced hereditary subgraphs

4.1 Introduction

The study of polynomial kernelization and, in particular, techniques for ruling out polynomial kernelizations has turned into one of the most well-studied directions in parameterized complexity [BDFH09, BJK14, DM12, DvM10, DLS09, FS11, HW12, Kra12]. In Chapter 3, we have already seen the techniques used to rule out polynomial kernels. The aim of this chapter is to explore the use of co-nondeterminism for the purpose of showing lower bounds for kernelization.

The only lower bound result based on a co-nondeterministic composition prior to this work was given by Kratsch [Kra12]. It is shown that the problem of finding an independent set *or* a clique of size at least k in a given graph does not admit a polynomial kernel with respect to k; we call the problem RAMSEY for its relation to Ramsey's theorem. The composition used in the lower bound proof [Kra12] relies on embedding instances of an improvement version of RAMSEY into a so-called host graph; intuitively, co-nondeterminism is used to find such a graph.

RAMSEY is a special case of the Π-INDUCED SUBGRAPH problem (defined below) whose parameterized complexity was determined by Khot and Raman [KR02]. This chapter seeks to develop kernelization lower bounds for Π-Induced Subgraph (when parameterized by the solution size k), hoping to find further applications of co-nondeterminism for kernelization lower bounds.

Finding induced Π-subgraphs. For a hereditary (i.e., closed under taking induced subgraphs) graph property Π, the Π-Induced Subgraph problem asks for the largest induced subgraph in the given graph G that belongs to the class Π. A classical result by Lewis and Yannakakis [LY80] asserts that Π-Induced Subgraph is NP-hard for any non-trivial (i.e., both Π and its complement are infinite) hereditary property Π.

Π-Induced Subgraph **Parameter(s):** k.

Input: A graph G and an integer k.

Question: Does there exist an induced subgraph of G on k vertices that belongs to Π?

Note that if Π contains all independent sets and all cliques, Π-Induced Subgraph is fixed-parameter tractable and admits a kernel of exponential size: By Ramsey's theorem, if G is too large, it contains a clique or an independent set of size k. If Π excludes both cliques and independent sets of a certain size then, by Ramsey's theorem, Π is finite and the problem is trivial. Khot and Raman [KR02] proved that for all other graph classes Π, Π-Induced Subgraph becomes W[1]-hard. Given this characterization, and the fact that *any* kernel would also imply fixed-parameter tractability, we study the existence of polynomial kernelizations when Π is restricted to contain all independent sets and cliques.

Our work. Regarding Π-Induced Subgraph we show that for most natural graph classes Π, including cographs, chordal, interval, split, perfect, and cluster graphs, there is no polynomial kernelization unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$; see Table 4.1 for an overview of some covered graph classes. This is proved by lower bounds for two classes of choices for Π which are established by a co-nondeterministic or-cross-composition and a parameterized reduction from Ramsey respectively.

As one tool for our compositions we establish a nice trick for allowing easier source problems

graph class	closed under embedding	has E-H property	forbids biclique	covered by Theorem
perfect, weakly chordal, cographs, comparability	Yes	Yes	No	5
chordal, interval, cluster, proper interval	No	Yes	Yes, $K_{2,2}$	6 and 7
split	No	Yes	Yes, $K_{2,2}$	7
claw-free	No	Yes	Yes, $K_{3,3}$	6 and 7
AT-free	Yes	open	No	5 (assuming E-H conjecture)

Table 4.1: A partial catalogue of graph classes covered by our results; all except split graphs are closed under disjoint union.

which should be of independent interest for other lower bounds. We show that for two general classes of problems, modeled after monotone and anti-monotone optimization problems, it suffices to start from *improvement versions*, where each instance comes with a guaranteed solution which is only off by one from the target value. For example, we define Improvement Π-Induced Subgraph as follows.

Improvement Π-Induced Subgraph

Input: A graph G, an integer k and a set $X \subseteq V(G)$ of size $k-1$ such that $G[X] \in \Pi$.

Question: Does there exist an induced subgraph of G on k vertices that belongs to Π?

Since the breakthrough paper of Bodlaender et al. [BDFH09] it has been known that improvement versions are useful source problems for deriving compositions. Unfortunately, this comes at a price: The framework requires to show NP-hardness of the improvement version; this may be straightforward, but it can also be "a tough nut" or outright impossible. By introducing co-nondeterminism also into this part of the picture we are able to show NP-hardness *under co-nondeterministic many-one reductions* (see Section 4.3 for a precise definition) and establish that this is sufficient for all existing variants of or-(cross-)compositions. More accurately, it suffices for lower bound frameworks that show $L \in \mathsf{coNP/poly}$ for some NP-hard L and hence get $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

The Erdős-Hajnal conjecture. Surprisingly, the question of kernelizations for Π-Induced Subgraph turns out to be related to the well-known Erdős-Hajnal conjecture about Ramsey bounds in hereditary graph classes. For an undirected graph G, let $\hom(G)$

denote the largest size of any clique or independent set of G. The well-known Ramsey's theorem asserts that $\hom(G) \geq \frac{1}{2}\log|V(G)|$ [ES35], and with high probability $\hom(G) = \mathcal{O}(\log|V(G)|)$ for random graphs $G(n, \frac{1}{2})$ [Erd47]. Clearly the randomized argument for the upper bound fails if we assume that G belongs to some fixed non-trivial hereditary property Π. Erdős and Hajnal conjectured in 1989 that indeed forbidding a fixed induced subgraph may change the behavior of $\hom(G)$ dramatically.

Conjecture 4.1 ([EH89])**.** *For every graph H there exists a constant $\varepsilon(H) > 0$ such that if G does not contain H as an induced subgraph, then $\hom(G) \geq |V(G)|^{\varepsilon(H)}$.*

On the one hand, a proof of this conjecture would give polynomial bounds for Ramsey numbers when *restricted to any hereditary class Π of graphs.* On the other hand, it implies exponential lower bounds for the minimum number of vertices required to force the existence of a Π subgraph of a certain size *in general graphs.*[1] The latter aspect is the one used in all of our results.

Related work. An alternative way of parameterizing Π-Induced Subgraph is to use the complement parameter, i.e., the number of vertices to be deleted to result in a graph in Π. Cai [Cai96] has shown that this problem is fixed-parameter tractable and admits a polynomial sized kernel when Π forbids a finite set of induced subgraphs. A complete characterization is open for hereditary Π with an infinite forbidden set, though parameterized results are known for specific properties, e.g., [Mar10, MS12, vBKMN10, FSV13, vtHV13].

Organization. In Section 4.3 we give a first example of how NP-hardness under co-nondeterministic many-one reductions suffices for OR-cross-compositions, using the example of Improvement Π-Induced Subgraph. In Section 4.4 we give two co-nondeterministic OR-cross-compositions that together rule out polynomial kernels for Π-Induced Subgraph for a wide selection of properties Π. In Section 4.5 we give a direct transformation from Ramsey to Π-Induced Subgraph, for certain further choices of Π. In Section 4.6 we

[1]In other words, one could define "Ramsey numbers" with respect to the class Π instead of the class of all independent sets and cliques, and, using the fact that Π-subgraphs contain "large" independent sets or cliques, show bounds very similar to those of the classical Ramsey theory.

generalize the result of Section 4.3 to *any* monotone or anti-monotone problem. We conclude in Section 4.7.

4.2 Preliminaries

Graphs and the Erdős-Hajnal property. We use standard graph notation, see e.g. [Die05]. We refer to [Gol80] for the definitions of some of the graph classes.

Definition 4.2. *A hereditary graph class* Π *has the* Erdős-Hajnal property *if there exists* $\varepsilon(\Pi) > 0$ *such that every* $G \in \Pi$ *has* $\hom(G) \geq |V(G)|^{\varepsilon(\Pi)}$. *I.e., every graph in* Π *has a clique or an independent set of size at least* $|V(G)|^{\varepsilon(\Pi)}$.

Lemma 4.1 (folklore)**.** *The class of perfect graphs, as well as any of its hereditary subclasses, has the Erdős-Hajnal property.*

Proof. For a perfect graph G the chromatic number $\chi(G)$ equals size of the largest induced clique $\omega(G)$. Either $\chi(G) = \omega(G) \geq |V(G)|^{1/2}$, or the largest color class is an independent set of size at least $|V(G)|/\omega(G) > |V(G)|^{1/2}$. □

Although Conjecture 4.1 is open, its statement is proven for many graphs H. One of the most important partial results is one due to Alon et al. [APS01].

Definition 4.3. *Let* G *be a graph and* $(H_x)_{x\in V(G)}$ *be a family of graphs, one for each vertex of* G. *We define the graph* $Embed(G; (H_x)_{x\in V(G)})$ *as the graph obtained from* G *by replacing each vertex* x *with the graph* H_x. *Formally,*

$$\begin{aligned} V(Embed(G; (H_x)_{x\in V(G)})) &= \{v(x,u) : x \in V(G), u \in V(H_x)\}, \\ E(Embed(G; (H_x)_{x\in V(G)})) &= \{v(x,u)v(x,w) : x \in V(G), uw \in E(H_x)\} \\ &\qquad \cup \{v(x,u)v(y,w) : xy \in E(G), u \in V(H_x), w \in V(H_y)\} \end{aligned}$$

We say that $Embed(G; (H_x)_{x\in V(G)})$ *is obtained* by embedding $(H_x)_{x\in V(G)}$ into G. *We say that a graph class* Π *is* closed under embedding *if for all* $G \in \Pi$ *and* $H_x \in \Pi$, *for* $x \in V(G)$, *the graph* $Embed(G; (H_x)_{x\in V(G)})$ *belongs to* Π.

Theorem 4.4 ([APS01]). *Let G be a graph and $(H_x)_{x\in V(G)}$ be a family of graphs. If the classes of G-free graphs and H_x-free graphs, for all $x \in V(G)$, satisfy the Erdős-Hajnal property, then the class of Embed$(G; (H_x)_{x\in V(G)})$-free graphs also satisfies the Erdős-Hajnal property.*

Corollary 4.2. *Let Π be a hereditary property that forbids $K_{\ell,\ell}$ for some integer ℓ. Then Π satisfies the Erdős-Hajnal property.*

Proof. First we show that $\overline{K_\ell}$-free graphs, for any $\ell \geq 1$, satisfy the Erdős-Hajnal property, by induction on ℓ. The case when $\ell = 1$ is trivial. For $\ell = 2$, the $\overline{K_\ell}$-free graphs are precisely cliques and they trivially satisfy the property. At the induction step, we use Theorem 4.4. We apply the theorem to Embed$(G; (H_x, H_y))$, where $G = \overline{K_2}$ on vertices x and y and with $H_x = \overline{K_1}$ and $H_y = \overline{K_{\ell-1}}$. It follows that $\overline{K_\ell}$-free graphs satisfy the Erdős-Hajnal property.

It is easy to see that K_2-free graphs satisfy the Erdős-Hajnal property because they are exactly the independent sets. Thus, using $K_{\ell,\ell} = \text{Embed}(K_2; (\overline{K_\ell}, \overline{K_\ell}))$ and Theorem 4.4, we get that $K_{\ell,\ell}$-free graphs satisfy the Erdős-Hajnal property. □

Many well-known graph classes are closed under embedding (see, e.g., [Lov83]).

Proposition 4.3. *Cographs, perfect graphs, permutation graphs, weakly chordal graphs, and AT-free graphs are closed under embedding.*

4.3 Co-nondeterminism and improvement versions

In this section we give a first example of how co-nondeterminism can also help relax the need for NP-hard source problems in cross-compositions (recall that this means hardness under standard Karp reductions). We show, using NP-hardness under polynomial-time co-nondeterministic many-one reductions, that, e.g., natural improvement versions of standard NP-hard problems can be used too (despite the fact that there are sometimes no NP-hardness proofs for those). We include here a proof for special case of Improvement Π-Induced Subgraph which suffices for our lower bounds for Π-Induced Subgraph; a more general result is given in Section 4.6.

Definition 4.5. *Let $L, L' \subseteq \Sigma^*$ be two languages. We say that a nondeterministic polynomial time algorithm $\mathcal{A}$ is a* co-nondeterministic many-one reduction *from L to L' if the following holds:*

1. *if $x \in L$, then on all computation paths $\mathcal{A}(x) \in L'$;*
2. *if $x \notin L$, then there exists a computation path with $\mathcal{A}(x) \notin L'$.*

We need the following easy fact.

Proposition 4.4. *Let L and L' be languages with $L' \in \mathsf{coNP/poly}$. If there is a co-nondeterministic many-one reduction from L to L' then $L \in \mathsf{coNP/poly}$.*

Proof. As $L' \in \mathsf{coNP/poly}$ there exists a co-nondeterministic polynomial-time machine M with polynomial advice for deciding membership in L'. We construct a $\mathsf{coNP/poly}$-machine with polynomial advice for deciding L. The machine will first perform the reduction to L' and on each ensuing computation path it will continue with a simulation of M applied to the output of the reduction.

It remains to describe the used advice string. Since the reduction can incur at most a polynomial blow-up of the initial input size n, we need only simulate M on inputs of size at most $p(n)$ where p is some fixed polynomial. The new machine gets as advice the concatenation of all advice strings for M for inputs of length up to $p(n)$ which too is polynomially bounded in n. $\square$

Observation 4.5. *Proposition 4.4 can be extended in the obvious way to co-nondeterministic reductions which are allowed polynomial advice of their own.*

Now we can give a general co-nondeterministic many-one reduction from Π-Induced Subgraph to Improvement Π-Induced Subgraph that works for any hereditary class Π with efficient membership test.

Lemma 4.6. *Let Π be a hereditary graph class for which membership can be tested in deterministic polynomial time. Then there exists a (polynomial-time) co-nondeterministic* *many-one reduction from* Π-Induced Subgraph *to* Improvement Π-Induced Subgraph.

Proof. Let (G, k) be an instance of Π-Induced Subgraph. The reduction will guess an integer $k' \in \{1, \ldots, k\}$ and a set X of $k' - 1$ vertices. Then it checks in deterministic polynomial time whether the subgraph induced by X is contained in Π, i.e., if $G[X] \in \Pi$. If this is not the case then a dummy Yes-instance of Improvement Π-Induced Subgraph is returned. Otherwise, the instance (G, X, k') is returned. This completes the description of the reduction.

Now, if (G, k) is a Yes-instance, then it is easy to see that each path returns a Yes-instance of Improvement Π-Induced Subgraph: The guessed set X is returned only (as part of (G, X, k')) if it induces a Π-graph in G. In all other cases a dummy Yes-instance is returned.

If (G, k) is a No-instance then consider the minimum value $k' \in \{1, \ldots, k\}$ such that (G, k') is a No-instance too; this value is guessed in one branch of the computation. Clearly G contains a Π-subgraph on $k' - 1$ vertices. Hence, one of the computation paths successfully guesses a subset X of $k'-1$ vertices such that $G[X] \in \Pi$. On this path the instance (G, X, k') is returned which can be easily seen to be a No-instance of Improvement Π-Induced Subgraph. □

Theorem 4.6. *Let* Π *be a non-trivial hereditary graph class for which membership can be tested in deterministic polynomial time. If there is a coNP-*OR*-cross-composition from* Improvement Π-Induced Subgraph *to* Π-Induced Subgraph *and* Π-Induced Subgraph *has a polynomial kernel or compression then* Π-Induced Subgraph $\in$ coNP/poly *and* NP $\subseteq$ coNP/poly.

Proof. It follows from Theorem 3.4 that the existence of both a coNP-OR-cross-composition from Improvement Π-Induced Subgraph to Π-Induced Subgraph and a polynomial kernelization for Π-Induced Subgraph implies that Improvement Π-Induced Subgraph $\in$ coNP/poly. By Proposition 4.4 and Lemma 4.6, we have Π-Induced Subgraph $\in$ coNP/poly and, by NP-hardness of Π-Induced Subgraph, that NP $\subseteq$ coNP/poly. □

4.4 Kernelization hardness by coNP-or-cross-compositions

In this section we give two co-nondeterministic OR-cross-compositions that apply for Π-Induced Subgraph for various choices of classes Π. We make use of our result from the previous section which permits us to use Improvement Π-Induced Subgraph as a source problem without proving (or requiring) NP-hardness; the (Karp) NP-hardness of Π-Induced Subgraph for nontrivial Π [LY80] suffices. Our first composition gives the following theorem.

Theorem 4.7. *Unless* NP ⊆ coNP/poly*, there is no polynomial kernelization or compression for* Π-Induced Subgraph *for any non-trivial hereditary graph class* Π *that is polynomially recognizable, contains all independent sets and cliques, is closed under embedding, and has the Erdős-Hajnal property.*

This theorem, for example, covers perfect graphs, and permutation graphs as these classes are closed under embedding due to Proposition 4.3; recall from Lemma 4.1 that all subclasses of the perfect graphs have the Erdős-Hajnal property. However, split graphs and chordal graphs are not covered by this theorem as they are not closed under embedding. See Table 4.1 in Section 4.1 for a partial catalogue of graph classes covered by our results.

By Theorem 4.6, it suffices to establish the following lemma to prove Theorem 4.7.

Lemma 4.7. *If* Π *is a hereditary graph class that is polynomially recognizable, contains all cliques and independent sets, is closed under embedding, and satisfies the Erdős-Hajnal property, then there exists a coNP-*OR*-cross-composition algorithm from* Improvement Π-Induced Subgraph *into* Π-Induced Subgraph.

We now proceed to the description of the coNP-OR-cross-composition algorithm for Improvement Π-Induced Subgraph towards proving Lemma 4.7. We start by preparing a host graph, similar to the case of the Ramsey problem [Kra12]. Fix a graph class Π satisfying the assumptions of Lemma 4.7 and let $\varepsilon > 0$ such that any $G \in \Pi$ satisfies $\hom(G) \geq |V(G)|^{\varepsilon}$.

Recall that the classical Ramsey number $R(\ell)$ is defined as the smallest integer such that any graph on $R(\ell)$ vertices contains an independent set or a clique of size ℓ. The following lemma has been shown in [Kra12].

Lemma 4.8 (Lemma 5.1 in [Kra12])**.** *For every integer $t > 3$ there exists an integer $\ell \in \{1, 2, \dots, \lceil 8 \log t \rceil\}$ such that $R(\ell + 1) > R(\ell) + t$.*

We define Π-Ramsey numbers as follows: for a positive integer ℓ, let $R_\Pi(\ell)$ be the smallest integer such that any graph on at least $R_\Pi(\ell)$ vertices contains an induced subgraph from Π of size ℓ. Note that this is well-defined, as all cliques and independent sets belong to Π, and $R_\Pi(\ell) \leq R(\ell)$. We show the following counterpart of Lemma 4.8.

Lemma 4.9. *There exists a constant $\delta(\Pi)$ that depends on the class Π only, such that for any integer $t > \delta(\Pi)$, there exists a positive integer $\ell \in \{1, 2, \dots, \lceil (8 \log t + 1)^{1/\varepsilon} \rceil\}$ such that $R_\Pi(\ell + 1) > R_\Pi(\ell) + t$.*

Proof. As Π satisfies the Erdős-Hajnal property with the exponent ε we have $R_\Pi(\ell) \geq R(\ell^\varepsilon)$: any graph that contains an induced subgraph from Π of size ℓ contains a clique or an independent set of size ℓ^ε. Recall the classical Erdős bound $R(\ell) \geq 2^{(\ell-1)/2}$; therefore $R_\Pi(\ell) \geq t^4$ for $\ell = \lceil (8 \log t + 1)^{1/\varepsilon} \rceil$. Now, let us assume that $R_\Pi(\ell + 1) \leq R_\Pi(\ell) + t$ for all $\ell \in \{1, 2, \dots, \lceil (8 \log t + 1)^{1/\varepsilon} \rceil\}$. Then $R_\Pi(\lceil (8 \log t + 1)^{1/\varepsilon} \rceil) \leq t(\lceil (8 \log t + 1)^{1/\varepsilon} \rceil - 1) + R_\Pi(1)$. Combining the two inequalities with the fact that $R_\Pi(1) = 1$, we get the following.

$$t^4 \leq t(\lceil (8 \log t + 1)^{1/\varepsilon} \rceil - 1) + 1$$

The above inequality is false for large enough values of t depending on ε, which in turn depends on the property Π. Hence, there exists a constant $\delta(\Pi)$ that depends on the class Π only, such that for any integer $t > \delta(\Pi)$, the above inequality does not hold, which gives us the desired contradiction. □

Now we describe the co-nondeterministic construction of a host graph, which will later be extended to a coNP-OR-cross-composition.

Lemma 4.10. *There exists a nondeterministic algorithm that, given an integer $t > \delta(\Pi)$, in time polynomial in t either answers* `FAIL` *or outputs the following*

- *an integer $\ell = \mathcal{O}(\log^{1/\varepsilon} t)$,*
- *a graph H and a family of sets $(A_x)_{x\in V(H)}$, $A_x \subseteq V(H)$, such that: $|V(H)| = t + o(t)$ and the graph H satisfies the following covering property: for each $x \in V(H)$ the set $A_x \subseteq V(H)$ is of size ℓ, $x \in A_x$, and $H[A_x] \in \Pi$.*

Furthermore, there exists a computation path where the H outputted above satisfies an additional property that H does not contain an induced subgraph of size $\ell + 1$ that belongs to Π.

Proof. We make a nondeterministic guess of a positive integer $\ell \in \{1, 2, \dots, \lceil(8\log t+1)^{1/\varepsilon}\rceil\}$ and a graph H_0 on at most $t(\lceil(8\log t + 1)^{1/\varepsilon}\rceil) + 1$ vertices. Observe that $\ell = \mathcal{O}(\log^{1/\varepsilon} t)$ and $|V(H_0)| = t^{\mathcal{O}(1)}$. By Lemma 4.9 there is an $\ell \in \{1, 2, \dots, \lceil(8\log t + 1)^{1/\varepsilon}\rceil\}$ such that $R_\Pi(\ell + 1) > R_\Pi(\ell) + t$. It can be easily verified that for the smallest such choice of ℓ we have $R_\Pi(\ell) + t \leq t(\lceil(8\log t + 1)^{1/\varepsilon}\rceil) + 1$. Hence, in at least one computation path, H_0 is a graph on $R_\Pi(\ell) + t < R_\Pi(\ell + 1)$ vertices which does not contain any induced subgraph from Π of size $\ell + 1$ (by the definition of $R_\Pi(\ell + 1)$).

Then we cut the graph H from H_0: start with $S = \emptyset$ and, while $|S| < t$, repeatedly guess a set $A \subseteq V(H_0 \setminus S)$ such that $H_0[A] \in \Pi$ and $|A| = \ell$ and take $S := S \cup A$. At each step, we verify whether $H_0[A] \in \Pi$: if not, we terminate and output `FAIL`. Note that if H_0 has (at least) $R_\Pi(\ell) + t$ vertices then, as long as $|S| < t$, we have $|V(H_0 \setminus S)| \geq R_\Pi(\ell)$ and there exists at least one set A of size ℓ such that $H_0[A] \in \Pi$. This guarantees that feasible sets A are found on at least one computation path. Finally, when $|S| \geq t$ we take $H = H_0[S]$; note that $t \leq |S| < t + \ell = t + o(t)$. $\square$

Proof of Lemma 4.7. Let $I_1, \dots, I_t$ be t instances of Improvement Π-Induced Subgraph of maximum size N. It is straightforward to efficiently partition such instances into $N^{\mathcal{O}(1)}$ equivalence classes: a) malformed instances, b) instances which are trivially No since k exceeds the number of vertices (which is bounded by N), c) remaining instances with equal value of k. Note that all triples (G, X, k) with $G[X] \notin \Pi$ are treated as malformed instances and can be sorted into the first equivalence class since Π is polynomially recognizable. Hence we may assume well-formed instances of the form $I_j = (G_j, X_j, k)$, with $k \leq N$ and

$G_j[X_j] \in \Pi$, since compositions for classes a) and b) are trivial. By possibly duplicating some instances, we may assume that t is larger than the constant $\delta(\Pi)$ given by Lemma 4.9 for the class Π.

We start by invoking the algorithm from Lemma 4.10 for the class Π and integer t, but modify its outputs appropriately. If the algorithm would return `FAIL`, we output a trivial YES-instance of Π-INDUCED SUBGRAPH instead.

Otherwise, we use the graph H and the integer ℓ to construct a graph G' by embedding one graph G_i into each vertex of H; as $|V(H)| = t + o(t)$, each graph G_i is embedded at least once. More formally, we take an arbitrary surjective function $\sigma : V(H) \to \{1, 2, \ldots, t\}$ and let $G' = \mathrm{Embed}(H; (G_{\sigma(x)})_{x \in V(H)})$. Set $k' = \ell(k-1)+1$. We return the instance (G', k') of Π-INDUCED SUBGRAPH. Clearly, as $\ell = \mathcal{O}(\log^{1/\varepsilon} t)$, we have $k' \leq (N + \log t)^{\mathcal{O}(1)}$ and the algorithm runs in nondeterministic polynomial time. We now verify its correctness.

Assume first that (G_i, X_i, k) is a YES-instance to IMPROVEMENT Π-INDUCED SUBGRAPH for some $1 \leq i \leq t$: let $Y \subseteq V(G_i)$, $|Y| = k$, $G_i[Y] \in \Pi$. Recall that in this case the coNP-OR-cross-composition algorithm should output a YES-instance in every computation path; this is clearly true if the algorithm of Lemma 4.10 fails to construct a graph H. Otherwise, let $x \in V(H)$ be such that $\sigma(x) = i$; recall that $x \in A_x \subseteq V(H)$, $|A_x| = \ell$, and $H[A_x] \in \Pi$. Let $Y_x = Y$ and $Y_y = X_{\sigma(y)}$ for $y \in A_x \setminus \{x\}$. Then $Y' = \{v(y,u) : y \in A_x, u \in Y_y\}$ is a set of size $k' = \ell(k-1)+1$ that induces in G' a graph isomorphic to $\mathrm{Embed}(H[A_x]; (G[Y_y])_{y \in A_x})$. As $H[A_x] \in \Pi$ and $G_{\sigma(y)}[Y_y] \in \Pi$ for $y \in A_x$, we infer that $G'[Y'] \in \Pi$ (since Π is closed under embedding) and (G', k') is a YES-instance for Π-INDUCED SUBGRAPH.

For the second case, assume that no graph G_i contains an induced subgraph of size k that belongs to Π. Let us focus on a computation path where a graph H and integer ℓ are generated such that H does not contain an induced subgraph from Π of size $\ell + 1$. We claim that in this computation path the generated instance (G', k') is a NO-instance for Π-INDUCED SUBGRAPH. Assume the contrary: let $Y' \subseteq V(G')$, $|Y'| = k'$, and $G'[Y'] \in \Pi$. Let $A = \{x \in V(H) : \exists_u : v(x,u) \in Y'\}$. Note that $H[A]$ is an induced subgraph of $G'[Y']$, thus $H[A] \in \Pi$. Since H does not contain an induced subgraph from Π of size $\ell + 1$, $|A| \leq \ell$. As $|Y'| = k' = \ell(k-1)+1$, by Pigeonhole Principle, there exists $x \in A$ such that

$Y = \{u \in V(G_{\sigma(x)}) : v(x,u) \in Y'\}$ is of size at least k. Moreover, $G_{\sigma(x)}[Y]$ is an induced subgraph of $G'[Y'] \in \Pi$, thus Y induces a subgraph in $G_{\sigma(x)}$ of size at least k that belongs to Π, a contradiction. $\square$

By a similar but slightly more technical coNP-OR-cross-composition we can prove a similar result for graph classes that are not necessarily closed under embedding, but instead exclude a certain biclique $K_{s,s}$; e.g., chordal graphs, which exclude $C_4 = K_{2,2}$.

Theorem 4.8. *Unless* NP $\subseteq$ coNP/poly*, there does not exist a kernelization algorithm with polynomial guarantee on the output size for* Π-INDUCED SUBGRAPH *for any non-trivial hereditary graph class* Π *that is polynomially recognizable, closed under disjoint union and excludes a certain biclique.*

The proof of Theorem 4.8 inspired the slightly more general lower bound result captured by Theorem 4.9 which is obtained by giving a polynomial parameter reduction from the RAMSEY problem to Π-INDUCED SUBGRAPH; this is showed in the following section, and Theorem 4.9 covers both split and chordal graphs as these classes forbid induced cycles of length 4 (i.e. $K_{2,2}$).

Nevertheless, we give the proof here since (apart from inspiring the mentioned PPT) it also introduces a few new tweaks to the co-nondeterministic approach that is a driving motivation behind this work. As a first step we need to improve slightly the host graph construction used for RAMSEY [Kra12]. The difference is that we can guarantee a cover by independent sets (as opposed to independent sets and cliques).

Lemma 4.11. *There is a (co-)nondeterministic algorithm such that, on input of any integer* $t > 1$*, the algorithm takes time polynomial in* t *and each computation path returns either* `FAIL` *or a graph* H *and an integer* $\ell = \mathcal{O}(\log t)$ *such that:*

1. *the vertices of* H *can be covered by independent sets of size* ℓ*;*
2. $t \le |V(H)| \le t + \mathcal{O}(\log t)$*;*
3. *on at least one computation path,* H *does not contain an independent set or a clique of size larger than* ℓ*.*

Proof. We follow the procedure used in Lemma 5.2 of [Kra12] to nondeterministically construct host graphs H, but we use the target size of $t' := 2t$ instead of t. That is, we first guess integers $\ell \in \{1, 2, \ldots, \lceil 8 \log t' \rceil\}$ and $T \in \{1, 2, \ldots, (\lceil 8 \log t' \rceil\} + 1)t'\}$ and a graph H_0 on T vertices. Second, we guess t' sets $A_i \subseteq V(H_0)$, $1 \le i \le t'$, each of size exactly ℓ. If there exists a set A_i that does not induce a clique nor an independent set in H_0, or if $\bigcup_{i=1}^{t'} A_i$ has less than t' vertices, then we return `FAIL`. Otherwise, let $\mathcal{A}^I$ and $\mathcal{A}^C$ be the family of these sets A_i that induce an independent set or a clique in H_0, respectively. Observe that, since $\bigcup_{i=1}^{t'} A_i$ has at least $t' = 2t$ vertices, either $\bigcup \mathcal{A}^I$ or $\bigcup \mathcal{A}^C$ is of size at least t. In the first case, pick any minimal subfamily $\mathcal{A}' \subseteq \mathcal{A}^I$ such that $\bigcup \mathcal{A}'$ is of size at least t, and return $H := H_0[\bigcup \mathcal{A}']$. Otherwise, pick any minimal subfamily $\mathcal{A}' \subseteq \mathcal{A}^C$ such that $\bigcup \mathcal{A}'$ is of size at least t, and return H being *the complement of* $H_0[\bigcup \mathcal{A}']$.

Clearly, $\ell = \mathcal{O}(\log t)$ and $t \le |V(H)| \le t + \ell = t + \mathcal{O}(\log t)$. Moreover, the family $\mathcal{A}'$ witnesses that the vertices of H can be covered by independent sets of size ℓ. Hence, it remains to argue about the last property promised in the lemma statement.

To this end, observe that, by Lemma 4.8, on one computation path we guess the integer ℓ with the following property: ℓ is the smallest positive integer such that $R(\ell+1) > R(\ell)+t'$. Furthermore, if this is the case, then we have $R(\ell) \le (\ell-1)t' + R(1)$ and in one computation path $T = R(\ell) + t' < R(\ell+1)$. Consequently, there exists a further computation path where the guessed graph H_0 does not contain any clique nor independent set of size $\ell + 1$. Let us focus on the computation path where ℓ, T and H_0 has been guessed as above.

Let $\mathcal{A}$ be the family of all sets $A \subseteq V(H_0)$ of size ℓ that induce a clique or an independent set in H_0. Observe that $H_0 \setminus \bigcup \mathcal{A}$ does not contain any clique or independent set on ℓ vertices. By the definition of $R(\ell)$, $|V(H_0)| - |\bigcup \mathcal{A}| < R(\ell)$ and, hence, $|\bigcup \mathcal{A}| > t'$. Consequently, there exists a choice of t' sets $A_i \in \mathcal{A}$, $1 \le i \le t'$, such that $|\bigcup_{i=1}^{t'} A_i| \ge t'$ and such a choice is made in at least one computation path. In this particular path, a graph H is returned. Since H is an induced subgraph of H_0, it does not contain any independent set nor clique of size larger than ℓ. This concludes the proof of the lemma. $\square$

Similarly as in the previous case, to prove Theorem 4.8 it suffices to show the following.

Lemma 4.12. *If* Π *is a non-trivial hereditary graph class that is polynomially recognizable,* *closed under disjoint union and excludes a certain biclique, then there exists a coNP-*OR*-cross-composition algorithm from* Improvement Π-Induced Subgraph *into* Π-Induced Subgraph*.*

Proof. Note that by being non-trivial and closed under disjoint union, the class Π must contain all independent sets. Moreover, by Corollary 4.2, the class Π fulfills the Erdős-Hajnal property. Let ε be the corresponding exponent, i.e., each $G \in \Pi$ contains an independent set or a clique of size $|V(G)|^{\varepsilon}$. Let $K_{s,s}$ be a biclique that is not in Π.

Let $I_1, \ldots, I_t$ be t instances of Improvement Π-Induced Subgraph. By use of an appropriate polynomial equivalence relation, it suffices to give a coNP-OR-cross-composition for the case where all instances are well-formed Improvement Π-Induced Subgraph instances of the form $I_j = (G_j, X_j, k)$, where G_j is a graph on n vertices. Recall that a triple (G, X, k) is *not* a well-formed instance if $G[X] \notin \Pi$; note that we assume that Π is polynomially recognizable. Furthermore, without loss of generality assume $t > 1$ (otherwise, duplicate the single input instance). We will describe a coNP-OR-cross-composition into one instance (G', X', k') of Π-Induced Subgraph. Let G'_i denote the graph consisting of c copies of G_i. The constant c will be fixed to be polynomially bounded in n and $\log t$ later.

The algorithm from Lemma 4.11 is extended to produce the desired reduction. We let it run on input t and modify the output. If the algorithm reports `FAIL` then we return a dummy Yes-instance of Π-Induced Subgraph. Otherwise, if it creates a host graph H on at least t vertices and an integer ℓ then we generate a graph G' by embedding a graph G'_i into each vertex of H (we copy instances if H has more than t vertices). We return the instance (G', k') of Π-Induced Subgraph, where $k' := c(k-1)\ell + c$. This completes the OR-cross-composition. It is easy to check that if c is polynomially bounded in n and $\log t$, the parameter value k' can be bounded polynomially in n and $\log t$, since $\ell = \mathcal{O}(\log t)$ by Lemma 4.11.

To show correctness let us first assume that at least one instance (G_i, X_i, k) is Yes. Thus G_i contains a Π-subgraph on k vertices. Hence, G'_i contains a Π-subgraph on ck vertices since Π is closed under disjoint union. Note that all other graphs G_j, for $j \neq i$, contain Π

subgraphs of size at least $k-1$, and G'_j graphs contain Π-subgraphs of size $c(k-1)$. Consider an independent set I of H of size ℓ, containing the vertex that we embedded G'_i into during the construction. It is easy to see that the Π-subgraphs of the corresponding graphs G'_j (those embedded into the vertices of I to get G') can be combined into one. Thus G' contains a Π-subgraph on at least $ck + (\ell - 1) \cdot c(k-1) = k'$ vertices.

Now, we check the case where all of the instances (G_i, X_i, k) are No. Consider a host graph H and integer ℓ such that H has a covering by independent sets of size ℓ but does not contain independent sets or cliques on more than ℓ vertices. Such a graph is guaranteed to be generated by the algorithm according to Lemma 4.11. It follows that H cannot contain an induced Π-subgraph of size larger than $\ell^{1/\varepsilon}$ since such a subgraph would contain an independent set or a clique larger than ℓ.

We assume for contradiction that the output instance is Yes and G' contains a Π-subgraph P with at least k' vertices. Clearly, P contains vertices from at most $\ell^{1/\varepsilon}$ graphs G'_i (in G'), since the corresponding vertices of H must form a Π-subgraph too (these vertices form an induced subgraph of $P \in \Pi$).

Consider the graphs G'_i such that P contains vertices from at least s copies of the graph G_i (recall that each G'_i consists of c disjoint copies of G_i). We observe that the vertices of H corresponding to those graphs G'_i must form an independent set: Otherwise, P would contain a $K_{s,s}$, subgraph, which is forbidden for Π. Hence, by the bound on independent sets in H, there are at most ℓ graphs G'_i from which P contains vertices from at least s copies of G_i.

Since, by assumption, no graph G_i contains a Π-subgraph on at least k vertices, it follows that the number of vertices of P is bounded by $\ell \cdot c(k-1) + \left(\ell^{1/\varepsilon} - \ell\right)(s-1)(k-1)$. The first term refers to the at most ℓ graphs G'_i from which P may contain vertices in all copies of G_i; the size of those subsets is limited to $k-1$ vertices per copy, under our assumption that all input instances are No. The second term refers to all other graphs G'_i in which at most $(s-1)$ copies of G_i can contribute at most $k-1$ vertices each. Now we fix the constant $c = \lceil sk\ell^{1/\varepsilon} \rceil$, so that

$$\ell \cdot c(k-1) + \left(\ell^{1/\varepsilon} - \ell\right)(s-1)(k-1) < \ell \cdot c(k-1) + c = k'.$$

Hence, the considered computation path returns a No-instance and c is polynomially bounded in n and $\log t$ for any fixed Π, as required. This completes the proof. □

Corollary 4.13. *Let Π be a non-trivial hereditary class of graphs with the properties as assumed in Theorem 4.8.* Π-Induced Subgraph *and* Improvement Π-Induced Subgraph *do not admit polynomial kernelizations or compressions with respect to the solution size k unless* NP ⊆ coNP/poly.

4.5 Kernelization hardness by transformation from RAMSEY

In this section we establish the following theorem which rules out polynomial kernelizations for Π-Induced Subgraph when Π excludes some biclique $K_{s,s}$.

Theorem 4.9. *Unless* NP ⊆ coNP/poly, *there does not exist a kernelization algorithm with polynomial guarantee on the output size for* Π-Induced Subgraph *for any non-trivial hereditary graph class Π that is polynomially recognizable, contains all independent sets, but excludes a certain biclique.*

We note that Theorem 4.9 is more general than Theorem 4.8, e.g., by including split graphs, but still does not cover all the cases of Theorem 4.7, e.g., the classes of perfect graphs and cographs are closed under embedding, but contain all bicliques.

We prove the theorem by a polynomial parameter transformation from the Ramsey problem. Let us recall the problem definition.

Ramsey **Parameter(s):** k.

Input: A graph G and an integer k.

Question: Does G contain an independent set or a clique of size k?

The following lemma, together with Theorem 3.6 and kernelization/compression hardness of Ramsey [Kra12], proves Theorem 4.9.

Lemma 4.14. *For any graph class Π that is polynomially recognizable, contains all independent sets, but excludes some biclique, there exists a polynomial parameter transformation that, given an instance (G, k) of* Ramsey*, outputs an equivalent instance (G', k') of* Π-Induced Subgraph*.*

Proof. Let $K_{s,s}$ be a biclique that is not in Π. Let G^2 be the join of G and its complement $\overline{G}$; that is, $V(G^2) = V(G) \uplus V(\overline{G})$ and $E(G^2) = E(G) \cup E(\overline{G}) \cup \{uw : u \in V(G), w \in V(\overline{G})\}$. We define the graph G' as a graph obtained from G^2 by embedding an independent set of size c into each vertex, where c is a parameter, polynomially bounded in k, that will be chosen later. In other words, $G' = \text{Embed}(G^2; (\overline{K_c})_{x \in V(G^2)})$. Let $k' = ck$. We claim that, if c is sufficiently large, the Ramsey instance (G, k) is equivalent to the Π-Induced Subgraph instance (G', k').

In one direction, let (G, k) be a Yes-instance to Ramsey. Then G^2 contains an independent set of size k, say $X \subseteq V(G^2)$. Then $X' = \{v(w, i) : w \in X, 1 \leq i \leq c\}$ is an independent set of size $k' = kc$ in G', and (G', k') is a Yes-instance to Π-Induced Subgraph.

In the other direction, let $X' \subseteq V(G')$ be a set of size k' such that $G'[X'] \in \Pi$. Let $Y = \{w \in V(G^2) : \exists_i v(w, i) \in X'\}$ and $Z = \{w \in Y : |\{1 \leq i \leq c : v(w, i) \in X'\}| \geq s\}$. The key observation is that since $G'[X']$ does not contain the biclique $K_{s,s}$ as an induced subgraph, Z is an independent set in G^2. Indeed, if $u, w \in Z$ and $uw \in E(G^2)$ then any s vertices of the form $v(w, i)$ together with any s vertices of the form $v(u, j)$ induce $K_{s,s}$ in G'. As G^2 is a join of G and $\overline{G}$, we infer that Z is wholly contained in $V(G)$ or wholly contained in $V(\overline{G})$.

If $|Z| \geq k$, we are done, as Z induces an independent set in G or in $\overline{G}$. Otherwise, as $|Z| < k$ but $|X'| = k'$, we infer that $|Y \setminus Z| > c/s$. So, either $|(Y \setminus Z) \cap V(G)| > c/(2s)$ or $|(Y \setminus Z) \cap V(\overline{G})| > c/(2s)$. We know by Lemma 4.2, there exists a constant $\varepsilon = \varepsilon(K_{s,s}) > 0$, such that for any $K_{s,s}$-free graph H (in particular, for any $H \in \Pi$), $\text{hom}(H) \geq |V(H)|^{\varepsilon}$, and hence Π satisfies the Erdős-Hajnal property with the exponent ε. Choosing $c = 2sk^{1/\varepsilon}$, we get either $|(Y \setminus Z) \cap V(G)| > k^{1/\varepsilon}$ or $|(Y \setminus Z) \cap V(\overline{G})| > k^{1/\varepsilon}$. Note that for any fixed Π, c and k' are polynomially bounded in k.

Now, $G[Y \setminus Z], \overline{G}[Y \setminus Z] \in \Pi$, as both of them are isomorphic to an induced subgraph of $G'[X']$. We also know that either $G[Y \setminus Z]$ or $\overline{G}[Y \setminus Z]$ has more than $k^{1/\varepsilon}$ vertices and hence contains a set X of size k that induces a clique or an independent set in either G or $\overline{G}$. Hence the Ramsey instance (G, k) is a Yes-instance. □

4.6 Generalizing Theorem 4.6 to monotone and anti-monotone problems

This section provides a generalization of Theorem 4.6 to further problems of similar structure, namely to problems that are monotone or anti-monotone. We define a general notion of improvement versions for such problems. Here we make use of the fact that membership in NP-languages is equivalent to the existence of a polynomially bounded certificate which can be efficiently verified. Furthermore, it is crucial that there is some way of obtaining a trivial starting solution in order to benefit from (anti-)monotonicity; intuitively, one should think of examples like having a trivial vertex cover using all vertices of the graph, or a trivial independent set containing no vertices (or just one). Let us point out, that this section should also be seen as a recipe for proving NP-hardness under co-nondeterministic many-one reductions for other types of problems.

The following definition of monotone and anti-monotone languages is motivated by similar notions for optimization problems. The decision versions of such problems will be special cases of our result. However, it is only required that the behavior on the second component is monotone or anti-monotone; there is no assumption about how to get certificates (or solutions) for larger, respectively smaller, values of k. Additionally, for technical reasons it is required that monotone languages can be efficiently decided when the second component exceeds a certain threshold (this would correspond to a trivial solution of high cost for a minimization problem, e.g., a vertex cover containing *all* vertices of the graph).

Definition 4.10 (monotone, anti-monotone)**.** *Let $L \subseteq \Sigma^* \times \mathbb{N}$. We say that L is* anti-monotone *if $(x, k) \in L$ implies $(x, k') \in L$ for all $k' \in \{0, \dots, k\}$ and there is a deterministic* *polynomial-time computable predicate $\Psi_0(x)$ for deciding membership of instances $(x, 0) \in \Sigma^* \times \mathbb{N}$.*

We say that L is monotone *if $(x, k) \in L$ implies $(x, k') \in L$ for all $k' \geq k$ and there exist a polynomial $p\colon \mathbb{N} \to \mathbb{N}$ and a deterministic polynomial-time computable predicate $\Psi(x, k)$ for deciding membership of instances $(x, k) \in \Sigma^* \times \mathbb{N}$ with $k > p(|(x, k)|)$.*

Definition 4.11 (improvement version)**.** *Let $L \subseteq \Sigma^* \times \mathbb{N}$ be a language in NPand let $\Phi \subseteq \Sigma^* \times \Sigma^* \times \mathbb{N}$ be a polynomial-time computable predicate such that $(x, k) \in L$ if and only if there exists a polynomially bounded witness y such that $\Phi(x, y, k)$ is true, formally:*

$$\forall (x, k) \in \Sigma^* \times \mathbb{N} : (x, k) \in L \Leftrightarrow \exists y : |y| = |(x, k)|^{\mathcal{O}(1)} \wedge \Phi(x, y, k).$$

1. *If L is anti-monotone then we define $L^- := L^-(L, \Psi_0, \Phi)$ by*

   $$L^- := \{(x, y, k) \mid |y| = |(x, k)|^{\mathcal{O}(1)} \wedge (x, k) \in L \wedge ((k = 0 \wedge \Psi_0(x)) \vee \Phi(x, y, k - 1))\},$$

   where Ψ_0 is chosen according to Definition 4.10.

2. *If L is monotone then we define $L^+ := L^+(L, p, \Psi, \Phi)$ by*

   $$L^+ := \{(x, y, k) \mid |y| = |(x, k)|^{\mathcal{O}(1)} \wedge (x, k) \in L \wedge ((k > p(|(x, k)|) \wedge \Psi(x, k)) \vee \Phi(x, y, k + 1))\},$$

   where $p\colon \mathbb{N} \to \mathbb{N}$ and Ψ are chosen according to Definition 4.10.

Intuitively, the improvement versions L^+ and L^- come with a certificate y which guarantees that $(x, k+1)$ or $(x, k-1)$ is contained in L respectively. For decision versions of optimization problems this corresponds to providing a solution which is only off by one (too large or too small) with respect to the solution value k that is asked for.

The following theorem, the main result of this section, establishes that these improvement versions can be used for composition-based lower bounds without being NP-hard. The crucial piece is their NP-hardness under co-nondeterministic many-one reductions (which we show); it is not hard to see that this is sufficient to apply any kernelization lower bound technique which aims to prove NP $\subseteq$ coNP/poly by showing $L \in$ coNP/poly: Indeed, since every language L' in NPreduces to L by a co-nondeterministic many-one reduction, we get $L' \in \mathsf{coNP/poly}$ by applying Proposition 4.4; this directly implies $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

Theorem 4.12. *Let $L \subseteq \Sigma^* \times \mathbb{N}$ be an NP-complete language.*

1. *If L is monotone and there is a coNP-*OR*-cross-composition of L^+ into a parameterized problem $\mathcal{Q}$ then $\mathcal{Q}$ admits no polynomial kernel or compression unless* $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.
2. *If L is anti-monotone and there is a coNP-*OR*-cross-composition of L^- into a parameterized problem $\mathcal{Q}$ then $\mathcal{Q}$ admits no polynomial kernel or compression unless* $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

Proof. We first show Part 2 of the theorem. Assume that L^- has a coNP-OR-cross-composition into $\mathcal{Q}$. If $\mathcal{Q}$ has a polynomial kernel or compression then, by Theorem 3.4, it follows that L^- is contained in $\mathsf{coNP/poly}$. Note that the consequence $\mathsf{NP} \subseteq \mathsf{coNP/poly}$ does not follow directly since L^- is not assumed to be NP-hard. Instead we show a co-nondeterministic many-one reduction of L to L^- which, by Proposition 4.4, implies that $L \in \mathsf{coNP/poly}$ and, hence, that $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

Let $(x, k) \in \Sigma^* \times \mathbb{N}$. If $k = 0$ then $(x, k) \in L$ if and only if $\Psi_0(x)$ holds, which can be checked in polynomial time. Then we correspondingly return a dummy YES- or NO-instance of L^-. (In fact, for the anti-monotone case we could simply return $(x, \epsilon, 0)$, where ϵ is the empty string.)

Otherwise, if $k \geq 1$, we proceed as follows. First, we check in deterministic polynomial time whether $\Psi_0(x)$ holds; if it does not then we return a NO-instance, otherwise we continue. Next, we guess an integer $k' \in \{1, \ldots, k\}$. Then, we guess a string y of length at most $|(x, k)|^{\mathcal{O}(1)}$ and check in deterministic polynomial time whether the predicate $\Phi(x, y, k' - 1)$ holds. If it does not hold then we return a dummy YES-instance. Otherwise, we return the instance (x, y, k').

Let us check correctness. If $(x, k) \in L$, then $(x, k') \in L$ for all $k' \in \{0, \ldots, k\}$. In particular we have $(x, 0) \in L$ and, thus, $\Psi_0(x)$ holds and we do not output a NO-instance. Hence, the output depends on the guessed string y and the outcome of $\Phi(x, y, k' - 1)$. If the latter does not hold then the reduction outputs a dummy Yes-instance. Otherwise it returns an instance (x, y, k') such that $\Phi(x, y, k' - 1)$ holds. In that case, it follows from $(x, k') \in L$ that $(x, y, k') \in L^-$ too.

Now, let us consider the case that $(x, k) \notin L$. In this case, it follows from anti-monotonicity of L that there is a minimum integer value k' such that $(x, k') \notin L$ but $(x, \hat{k}) \in L$ for all non-negative integers $\hat{k} < k'$. If $k' = 0$, then $(x, 0) \notin L$ and our check of $\Psi_0(x)$ will lead to the output of a No-instance. Let us consider the more interesting case that $k' > 0$. We know that $(x, k' - 1) \in L$ and consequently the reduction will correctly guess a certificate y such that $\Phi(x, y, k' - 1)$ holds on at least one computation path. On that path the returned instance (x, y, k') can be easily seen not to be contained in L^-.

Thus our reduction correctly reduces L to L^- and it is easy to see that it can be performed in (nondeterministic) polynomial time. Thus, by Proposition 4.4, we have $L \in \mathsf{coNP/poly}$ and $\mathsf{NP} \subseteq \mathsf{coNP/poly}$. This completes Part 2.

The proof for Part 1 essentially works along the same lines. We will only sketch the small differences: Instead of $(x, 0)$, the trivial instances are all those instances (x, k) where $k > p(|(x, k)|)$. For those instances the reduction can determine the correct answer by computing $\Psi(x, k)$ followed by the output of a dummy Yes- or No-instance. All other instances have $k \leq p(|(x, k)|)$. For such an instance to be Yes, it is necessary that $(x, p(|(x, k)|) + 1)$ is Yes. The latter can be checked efficiently via Ψ, and we return a dummy No-instance if it does not hold. Afterwards, it suffices to guess k' from $\{k, \ldots, p(|(x, k)|)\}$ and to proceed analogously to the anti-monotone case. $\square$

4.7 Conclusion

We have studied further applications of co-nondeterminism for obtaining lower bounds for kernelizations, using Π-Induced Subgraph as our target problem. Our results are twofold. First, we have obtained two co-nondeterministic cross-compositions for Π-Induced Subgraph for certain properties Π, which are in style of those for the Ramsey problem [Kra12]. Additionally, a direct polynomial parameter transformation from Ramsey to Π-Induced

SUBGRAPH (for certain Π) covers further cases; see Table 4.1 for an overview. Note that if the kernelization hardness result holds for a class Π, then it also holds for the class that contains the complement of each of the graphs in Π. It would be interesting to know whether there are any non-trivial hereditary properties Π for which Π-INDUCED SUBGRAPH has a polynomial sized kernel. We conjecture that there are none.

Second, motivated by the necessity of proving NP-hardness of IMPROVEMENT Π-INDUCED SUBGRAPH as our source problem, we came up with a way of avoiding that issue entirely: We showed in general how to use such improvement versions of NP-hard problems without requiring the problems themselves to be NP-hard. Instead, it can be showed that all such problems are NP-hard under *co-nondeterministic many-one reductions* and that this suffices for all applications of OR-(cross-)compositions. More generally, it applies for any lower bound strategy that tries to prove $\mathsf{NP} \subseteq \mathsf{coNP/poly}$ by proving $L \in \mathsf{coNP/poly}$ for some NP-hard language L. Furthermore, this is not limited to improvement versions, but applies whenever we have only co-nondeterministic (rather than Karp) NP-hardness. We believe that this may simplify future lower bound proofs in the same way as it helped our kernelization hardness results for Π-INDUCED SUBGRAPH.

Chapter 5

Parameterized Algorithms for Max Colorable Induced Subgraph problem on Perfect Graphs

5.1 Introduction

The focus of this chapter is the Max q-Colorable Induced Subgraph problem, with a special focus on co-chordal graphs and perfect graphs. The problem can be looked as an instance of Π-Induced Subgraph where the class Π is the set of q-colorable graphs. Our results are of parameterized flavor, involving both FPT algorithms and lower bounds for polynomial kernels.

Before we can describe our results, we establish some basic notions. A graph $G = (V, E)$ is called *q-colorable* if there is a coloring function $f : V \to [q]$ such that $f(u) \neq f(v)$ for any $(u, v) \in E$. Equivalently, a graph is q-colorable if its vertex set can be partitioned into q independent sets. The Max q-Colorable Induced Subgraph problem asks for a maximum induced subgraph that is q-colorable, and the decision version, p-mcis, may be stated as follows:

p-Max q-Colorable Induced Subgraph (p-mcis) **Parameters:** ℓ

Input: An undirected graph $G = (V, E)$ and positive integers ℓ and q.

Question: Does there exist $Z \subseteq V$, $|Z| \geq \ell$, such that $G[Z]$ is q-colorable?

We will sometimes be concerned with the problem above for *fixed values of* q, and to distinguish this from the case when q is a part of the input, we use p-q-mcis to refer to the version where q is fixed. The problem is clearly NP-complete on general graphs as for $q = 1$ this corresponds to Independent Set problem. Yannakakis and Gavril [YG87] showed that this problem is NP-complete even on split graphs (which is a proper subset of perfect graphs, chordal graphs and co-chordal graphs). However, they showed that p-q-mcis is solvable in time $n^{\mathcal{O}(q)}$ on chordal graphs. A natural question, therefore, is whether the problem admits an algorithm with running time $f(q) \cdot n^{\mathcal{O}(1)}$ on chordal graphs, or even on split graphs. This question was our main motivation for looking at p-mcis on special graph classes like co-chordal and perfect graphs.

Our results and related work. Most of the "induced subgraph problems" are known to be W[1]-hard parameterized by the solution size on general graphs by a generic result of Khot and Raman [KR02]. In particular this also implies that p-mcis is W[1]-hard parameterized by the solution size on general graphs. Observe that Independent Set is essentially p-mcis with $q = 1$. There has been also some study of parameterized complexity of Independent Set on special graph classes [DLMR10, RS08]. Yannakakis and Gavril [YG87] showed that p-mcis is NP-complete on split graphs and Addario-Berry et al. [ABKK$^+$10] showed that the problem is NP-complete on perfect graphs for every fixed $q \geq 2$.

We observe in passing that the known NP-completeness reduction given in [YG87] implies that p-mcis when parameterized by q alone is W[2]-hard even on split graphs. The main results in this chapter are two randomized FPT algorithms for p-mcis and a complementary lower bound, which establishes the non-existence of a polynomial kernel under standard complexity-theoretic assumptions.

Our first algorithm runs in time $(2e)^{\ell}(n + \#\alpha(G))^{\mathcal{O}(1)}$ where $\#\alpha(G)$ is the number of maximal independent sets of the input graph and the second algorithm runs in time

$\mathcal{O}(6.75^{\ell+o(\ell)}n^{\mathcal{O}(1)})$ on graph classes where the maximum independent set of an induced subgraph can be found in polynomial time. The first algorithm is efficient when the input graph contains only polynomially many maximal independent sets; for example on split graphs and co-chordal graphs. The second algorithm is efficient for a larger class of graphs, for example perfect graphs, because it only relies on an efficient procedure for finding a maximum independent set (although this comes at the cost of slightly larger base of the exponent).

We also describe de-randomization procedures for our algorithms. While the derandomization technique for the first algorithm is standard, to derandomize the second algorithm we use the idea of (n, p, q)-separating families, introduced in [FLS14]. Further, we show that unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$, the problem does not admit polynomial kernel even on split graphs. Also, on perfect graphs, we show that the problem does not admit a polynomial kernel even for fixed $q \geq 2$, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

5.2 Preliminaries and Definitions

We first recall the definition of embedding of a graph into another from the previous chapter.

Definition 5.1. *Let $G = (V, E)$ and $H_x = (V_x, E_x)$ for $x \in V$ be graphs. We define the graph $G' = Embed(G; (H_x)_{x \in V})$ as the graph obtained from G by replacing each vertex x with the graph H_x . Formally, $V(G') = \{u_x | x \in V,\ u \in V_x\}$ and $E(G') = \{(u_x, v_x) | x \in V, (u, v) \in E_x\} \cup \{(u_x, v_y) | (x, y) \in E,\ u \in V_x,\ v \in V_y\}$.*

We say that the graph $Embed(G;\ (H_x)_{x \in V})$ is obtained by embedding $(H_x)_{x \in V}$ into G. We say that a graph class Π is closed under embedding if whenever $G \in \Pi$ and $H_x \in \Pi$ for all $x \in V(G)$, then the graph $Embed(G;\ (H_x)_{x \in V(G)})$ belongs to Π. It is known that perfect graphs are closed under embedding [Lov83].

Definition 5.2. *Let G be a graph and $E' \subseteq E(G)$. We define the graph $Triangular(G;\ E')$ as adding vertices x_e and edges (x_e, u), (x_e, v) for all $(u, v) = e \in E'$ to the graph G.*

If $G = (V, E)$ is a perfect graph and $E' \subseteq E$, then $Triangular(G;\ E')$ is also a perfect graph (the proof of this statement is implicit in the proof of Theorem 5.9).

A *Steiner Tree* for a set $T \subseteq V(G)$ is a connected subgraph of G contaning T. We will be using the known algorithm for Steiner Tree problem defined as follows.

Steiner Tree **Parameter(s):** $|T|$

Input: An undirected graph G, set $T \subseteq V(G)$ and an integer k.

Question: Does there exist a connected subgraph of G containing T with at most k additional vertices?

5.2.1 Derandomization Techniques

In this subsection, we will mention some derandomization techniques, namely families of perfect hash functions, (a, b)-separating families and (n, p, q)-separating collections.

Definition 5.3 ([NSS95])**.** *A family H of functions from $[n]$ to $[l]$ is called an (n, k, l)-family of perfect hash functions if for all $S \in \binom{[n]}{k}$, there is an $h \in H$ which is one-to-one on S.*

We need the following result regarding (n, k, k)-family of perfect hash functions.

Theorem 5.4 ([NSS95])**.** *There is a deterministic algorithm with running time $\mathcal{O}(e^k k^{\mathcal{O}(\log k)} n \log n)$ that constructs an (n, k, k)-family of perfect hash functions $\mathcal{F}$ such that $|\mathcal{F}| = e^k k^{\mathcal{O}(\log k)} \log n$.*

Definition 5.5 ([FLS14])**.** *An (n, x, y)-separating family over a universe U of size n is a family of sets $\mathcal{F}$ over U such that for all $A \in \binom{U}{x}$ and $B \in \binom{U \setminus A}{y}$, there exists $F \in \mathcal{F}$ such that $A \subseteq F$ and $B \cap F = \emptyset$.*

Theorem 5.6 ([FLS14])**.** *There is a deterministic algorithm with running time $\binom{x+y}{x} 2^{o(x+y)} n \log n$ that constructs an (n, x, y)-separating family $\mathcal{F}$ such that $|\mathcal{F}| = \binom{x+y}{x} 2^{o(x+y)} \log n$.*

5.3 FPT Algorithms

In this section we design two randomized algorithms for p-MCIS. The first algorithm requires a subroutine that enumerates all maximal independent sets (MIS) in the input graph and this algorithm is useful when the input graph has polynomially many maximal independent sets. We can derandomize this algorithm using a (n, ℓ, ℓ)-family of perfect hash functions.

The second algorithm requires a subroutine which computes a *maximum* independent set (or an independent set of particular size $\ell' \leq \ell$) of any induced subgraph of the input graph. Thus, this algorithm is FPT on all graph classes for which Independent Set is either polynomial time solvable or FPT parameterized by the solution size. We derandomize this algorithm using the (n, p, q)-separating falimies.

Notice that the second algorithm is less demanding than the first: we only need to find a largest independent set, rather than enumerating all maximal ones. Thus the second algorithm solves the problem for a larger class of graphs than the first, however, as we will see, the running time is compromised in the sense that the base of the exponent is slightly increased. In particular, this is why the second algorithm doesn't render the first obsolete. The first can be thought of as a more efficient algorithm when the class of graphs was restricted further.

5.3.1 Algorithm based on enumerating Maximal Indepenent Sets

Let $\#\alpha(G)$ denote the number of maximal independent sets of G. In this section we give a randomized algorithm with one sided error for p-MCIS that uses all the maximal independent sets in the graph, runs in time $2^{\ell}(|V(G)| + \#\alpha(G))^{\mathcal{O}(1)}$, and gives the correct answer with probability at least $e^{-\ell}$. The error is one-sided: if the input instance is No instance, then the algorithm will output No always. We first state the result about enumerating all maximal independent sets of a graph.

Lemma 5.1 ([TIAS77]). *There exists an algorithm for generating all maximal independent sets of an n-vertex m-edge graph G in time $\mathcal{O}(mn \cdot \#\alpha(G))$, where $\#\alpha(G)$ is the number*

```
**Input**:    A graph $G = (V, E)$ and positive integers $\ell$, $q$
**Output**: YES, if there exists $S \subseteq V$, $|S| = \ell$ and $G[S]$ is $q$-colorable, NO otherwise.
1 Using the algorithm of Lemma 5.1, enumerate all maximal independent sets in $G$.
  Let $M = \{m_1, m_2, \ldots, m_t\}$ be the set of all maximal independent sets.
2 Construct a split graph $G' = (V \uplus M, E' = \{(v, m_i) | \; m_i \in M, \; v \in V \cap m_i\})$, where
  $G'[M]$ is a clique.
3 Color each vertex in $V$ with a color from an $\ell$-sized set of colors uniformly at random.
4 Merge all vertices in each color class into a single vertex. Formally, replace each color
  class $C_i$ by a single vertex $c_i$, and let $N(c_i) = \{u \mid \exists v \in C_i, (u, v) \in E'\}$. Let the
  graph after contraction be $G^* = (C \uplus M, E^*)$.
5 If there is a set of $q$ vertices in $M$ which dominates $C$, then output YES, otherwise
  output NO.
```
Algorithm 1: An Algorithm for p-MCIS based on enumerating MIS.

of maximal independent sets in the graph G.

Now we are ready to prove the main lemma of this subsection.

Lemma 5.2. *If (G, ℓ, q) is a* YES *instance of p-*MCIS*, then Algorithm 1 will output* YES *with probability at least e^{-l}, otherwise Algorithm 1 will output* NO *with probability* 1*. Furthermore, on input (G, ℓ, q), Algorithm 1 runs in time $2^{\ell}(|V(G)| + \#\alpha(G))^{\mathcal{O}(1)}$.*

Proof. We first show the correctness of the algorithm whenever the output YES. Suppose Algorithm 1 outputs YES. Then there exist q vertices in M that dominates all vertices in C which implies at least one vertex in each color class that is dominated by one or more of these q vertices. In particular, there exists a subset $T \subseteq V$ with ℓ vertices and a subset $S \subseteq M$ with q vertices, such that S dominates T. We argue that $G[T]$ is the desired q-colorable subgraph. Let $T := \{v_1, v_2, \ldots, v_\ell\}$. For each v_i, let $c(v_i)$ be the smallest j for which v_i is dominated by m_j. Notice that c defines a partition of T into q sets. For all $1 \leq j \leq q$, it is clear that $c^{-1}(j)$ is a subset of some maximal independent set, and hence the proposed partition is a proper coloring. Therefore, (G, ℓ, q) is a YES instance of p-MCIS.

We now argue the probability that the algorithm finds a solution given that the input is a YES instance. Let (G, ℓ, q) be a YES instance of p-MCIS, and let $T \subseteq V$ with $|T| = \ell$, be a solution. When we randomly color the vertices, each vertex in T will get different colors with probability $\frac{\ell!}{\ell^\ell} \geq e^{-\ell}$. If T gets different colors then there exist q sets in M which dominate C because there exists a maximal independent set that contains each color class in $G[T]$ (since $G[T]$ is q-colorable). Hence Algorithm 1 will output YES with probability at least e^{-l}.

Now we argue the running time bound of the algorithm. Let $n = |V(G)|$ and $m = |E(G)|$. Step 1 takes time $\mathcal{O}(mn \cdot \#\alpha(G))$ due to Lemma 5.1 and steps 2—4 takes time $\mathcal{O}((\#\alpha(G))^2 \cdot n)$. To find a q sized set from M to dominate C in Step 5, we run a STEINER TREE algorithm on the graph G^* with C given as the set of terminals. We claim that there is a q sized set in M which dominates C exists if and only if there exists a Steiner Tree using at most q additional vertices to connect the terminal set C. If there is a Steiner tree for C with at most q additional vertices $\{s_1, \ldots, s_q\}$ from M, then notice that the non-terminal vertices in the Steiner Tree form a dominating set for C. On the other hand, if there is a set $M' \subseteq M, |M'| \leq q$ such that M' dominates C, then $G[M' \cup C]$ form a Steiner tree for C. Since finding the optimal Steiner Tree in an N-vertex graph with k terminals can be done in $2^k N^{\mathcal{O}(1)}$ time [BHKK07], we have that the last step of the algorithm runs in time $2^{\ell}(n + \#\alpha(G))^{\mathcal{O}(1)}$. Hence the claimed running time follows. □

We can boost the success probability to a constant by executing Algorithm 1 e^{ℓ} times, in which case the success probability will be at least $(1 - e^{-\ell})^{e^{\ell}} \geq \frac{1}{e}$. This, when combined with Lemma 5.1 gives us the following theorem.

Theorem 5.7. *We can solve p-MCIS with constant success probability in $(2e)^{\ell} n^{\mathcal{O}(1)}$ time on an n-vertex graph class where the number of maximal independent sets is bounded by a polynomial in n.*

It is easy to see that we can derandomize the algorithm using a (n, ℓ, ℓ)-family of perfect hash functions (see Theorem 5.4) to obtain a deterministic algorithm with running time $(2e)^{\ell} \ell^{\mathcal{O}(\log \ell)} n^{\mathcal{O}(1)}$ for p-MCIS on graph classes for which the number of maximal independent sets is bounded by a polynomial in n. Since the number of maximal cliques in chordal graphs with n vertices is bounded by n, the number of independent sets in co-chordal graphs are bounded by n. We also know that complement of split graphs are split and hence co-chordal. We therefore have the following corollary:

Corollary 5.3. *p-MCIS can be solved in time $(2e)^{\ell} \cdot \ell^{\mathcal{O}(\log \ell)}|V(G)|^{\mathcal{O}(1)}$ on co-chordal graphs and split graphs.*

5.3.2 Algorithm based on finding a Maximum Independent Set

In Algorithm 2, we describe a randomized FPT algorithm which succeeds with probability greater than $1/4$ on graph classes where MAXIMUM INDEPENDENT SET can be solved in polynomial time. Let $S \subseteq V(G)$ be such that $G[S]$ is an induced subgraph of G of size ℓ that is q-colorable. Let $\mathcal{S} = \{S_1, S_2, \ldots, S_{q'}\}$ be the set of color classes of $G[S]$ where $q' \leq q$. We say that $G[S]$ is *compatible* with a partition $P = \{\ell_1, \ell_2, \ldots, \ell_{q'}\}$ of ℓ into q' parts if for each S_i, there exists an injection f from $\mathcal{S}$ to P such that $f(S_i) = |S_i|$.

```
  Input:  A graph $G = (V, E)$ and positive integers $\ell$, $q$
  Output: YES, if there exists $S \subseteq V$, $|S| = l$ and $G[S]$ is $q$-colorable, NO otherwise.
1 Algorithm q-Colorable($G, \ell, q$)
2 Generate the set $\mathcal{P}$ of all the partitions of $\ell$ having most $q$ parts
3 for each $P \in \mathcal{P}$ do
4     if Partition-q-colorable($G, \ell, q, P$) returns YES then
5         return YES
6     end
7 end
8 return NO
```

Algorithm 2: An Algorithm for p-MCIS based on finding maximum IS.

Lemma 5.4. *If (G, ℓ, q) is a* YES *instance of p-MCIS, then Algorithm 2 will output* YES *with probability greater than $1/4$, otherwise Algorithm 2 will output* NO *with probability 1. Algorithm 2 runs in time $\mathcal{O}(8^{\ell} 2^{\sqrt{\ell}} n^{\mathcal{O}(1)})$ on graph classes where* MAXIMUM INDEPENDENT SET *can be solved in polynomial time.*

Proof. We first show that if Algorithm 2 outputs YES for (G, ℓ, q), then (G, ℓ, q) is indeed a YES instance. Since Algorithm 2 outputs YES only when the procedure Partition q-colorable outputs YES, it is sufficient to show that if Partition q-colorable outputs YES for an instance (G, ℓ, q, P), then (G, ℓ, q) is a YES instance of p-MCIS as well.

We do so by doing an induction on q. The base case is when $q = 1$, and then $\mathcal{P} = \{\ell\}$. In this case, the procedure Partition q-colorable outputs YES in step 5 when the maximum independent set in G has size at least ℓ and hence (G, ℓ, q) is a YES instance. Now, for $q \geq 2$,

```
Input:   A graph $G = (V, E)$, positive integers $\ell$, $q$ and a partition $P$ of $\ell$ into at
         most $q$ parts
Output:  YES, if there exists $S \subseteq V$, $|S| = \ell$, $G[S]$ is $q$-colorable and $G[S]$ is
         compatible with $P$, NO otherwise.
1  Procedure Partition-q-colorable($G, \ell, q, P$)
2  Let $P = \{\ell_1, \ell_2, \ldots, \ell_t\}$ where $\sum_{i \in t} \ell_i = \ell$ and $t \leq q$.
3  if $q = 1$ then
4      if the maximum independent set of $G$ has size at least $\ell$ then
5          return YES
6      else
7          return NO
8      end
9  end
10 if there exists $i$ such that $\ell_i \geq \ell/2$ then
11     Let $P_1 = \{\ell_i\}$, $P_2 = P \setminus \{\ell_i\}$
12     for $3 \cdot 2^l$ times do
13         Choose some $V' \in 2^V$ with uniform probability.
14         $G_1 := G[V']$ and $G_2 = G[V \setminus V']$
15         Find the maximum sized independent set, say $S$, in $G_1$
16         if $|S| \geq \ell_i$ and Partition-q-colorable($G_2, \ell - \ell_i, q - 1, P_2$) returns YES
           then
17             return YES
18         end
19     end
20 else
21     Divide $P$ into two parts $P_1$ and $P_2$ such that
       $\ell/3 \leq (\sum_{\ell_i \in P_1} \ell_i), (\sum_{\ell_i \in P_2} \ell_i) \leq 2\ell/3$
22     Let $\ell' := (\sum_{\ell_i \in P_1} \ell_i)$
23     for $3 \cdot 2^l$ times do
24         Choose some $V' \in 2^V$ with uniform probability.
25         $G_1 := G[V']$ and $G_2 = G[V \setminus V']$.
26         if Partition-q-colorable($G_1, \ell', |P_1|, P_1$) returns YES and
           Partition-q-colorable($G_2, \ell - \ell', |P_2|, P_2$) returns YES then
27             return YES
28         end
29     end
30 end
```
Algorithm 3: A procedure for p-MCIS based on finding maximum IS.

the procedure Partition q-colorable outputs YES either in Step 18 or in Step 27. If it outputs YES in Step 18, then the recursive call `Partition-q-colorable(`$G_2, \ell - |S|, q-1, P_2$`)` must have returned YES. Hence, because of the induction hypothesis, G_2 has a $q-1$ colorable subgraph of size $\ell - |S|$. This subgraph, when combined with the set S gives us an ℓ sized q-colorable subgraph of G and hence (G, q, ℓ) is a YES instance of p-MCIS. The other case can be shown similarly.

Now, to prove the statement of the lemma, it suffices to show if (G, ℓ, q) is a YES instance, then the algorithm outputs YES with probability greater than $1/4$, since we have already shown that when Algorithm 2 outputs YES, then (G, ℓ, q) is indeed a YES instance. Let (G, ℓ, q) be a YES instance of p-MCIS, $S \subseteq V, |S| = \ell$ be the solution set and $f : S \to [q]$ be a fixed proper coloring of $G[S]$. Let the color classes in q-coloring of S be $S_1, S_2, \dots S_t$, where $t \leq q$. Since we generate the set $\mathcal{P}$ of all the partitions of ℓ into at most q parts, it is easy to see that there exists $P \in \mathcal{P}$ such that $G[S]$ is compatible with P. Hence, the algorithm makes an error if and only if the procedure Partition q-colorable makes an error. Note that we are only looking for proving this claim when the procedure Partition q-colorable makes an error for a YES instance. That is, we will only be looking at a YES instances of p-MCIS where the the q-colorable subgraph $G[S]$ of size ℓ is compatible with the partition P provided to the the procedure.

Let p_ℓ be the probability that procedure Partition q-colorable gives wrong answer for a YES instance (G, ℓ, q, P). Using induction we will show that $p_\ell \leq 1/4$. The base case is when ℓ is 1, and there the procedure returns YES with probability 1. Now to proceed through the induction, we assume that for all $\ell' < \ell$, $p_{\ell'} < 1/4$. Now we do a case analysis on the partition $P = \{\ell_1, \ell_2, \dots, \ell_t\}$, where $\sum_{i \in t} \ell_i = \ell$ and $t \leq q$.

Case 1. There exists i such that $\ell_i \geq \ell/2$. Let S_j be the color class in $G[S]$ such that $|S_j| = \ell_i$. Such a color class exists because $G[S]$ is compatible with P. We call the two-partitioning of the graph in steps 13 and 14 of the algorithm *good* if $S_j \subseteq V'$ and $(S \setminus S_j) \subseteq (V \setminus V')$. Such a partition is achieved with probability $2^{-\ell}$. After that, the procedure finds the maximum independent set of G_1 (the part which contains S_j) with probability one. After a good partitioning, the only way the procedure Partition q-colorable can make an error is the case when the recursive call makes an error. We claim that the error probability, p_ℓ, is bounded by the following.

$$(1 - 2^{-\ell} + 2^{-\ell} p_{\lfloor \ell/2 \rfloor})^{3 \cdot 2^\ell}$$

This is because with probability $1 - 2^{-\ell}$ the partitioning is not good, and $2^{-\ell} p_{\lfloor \ell/2 \rfloor}$ is upper bound for the probability when the partitioning is good, but the recursive call makes an error. This gives us, by applying induction hypothesis, $p_\ell \leq (1 - 2^{-\ell} + 2^{-\ell}/4)^{3 \cdot 2^\ell} < (1 - 2^{-\ell} + 2^{-(\ell+1)})^{3 \cdot 2^\ell} = (1 - 2^{-(\ell+1)})^{\frac{3}{2} 2^{\ell+1}} \leq e^{-3/2} < 1/4$.

Case 2. There does not exist i such that $\ell_i \geq \ell/2$. In this case, we can divide the ℓ_i's into two parts such that sum of the ℓ_i's in both parts is between $\ell/3$ and $2\ell/3$. We call those partitions P_1 and P_2. Now, we call a two-partition of the graph in steps 24 and 25 *good* if all the S_i's corresponding to the ℓ_i's in P_1 are in the same partition, and all the remaining S_i's are in the other partition. Clearly, this happens with probability $2^{-\ell+1}$. Now, the procedure can make an error if either the partitioning is not good, or if one of the recursive calls makes an error. Hence, the error probability, p_ℓ, is bounded by

$$(1 - 2^{-\ell+1} + 2^{-\ell+1} \cdot 2p_{\lfloor 2\ell/3 \rfloor})^{3 \cdot 2^\ell}$$

because with probability $1 - 2^{-\ell}$ the partitioning is not good, and $2^{-\ell+1} \cdot 2p_{\lfloor 2\ell/3 \rfloor}$ is upper bound for the probability when the partitioning is good, but one of the recursive call makes an error. By induction, $p_\ell \leq (1 - 2^{-\ell+1} + 2 \cdot 2^{-\ell+1}/4)^{3 \cdot 2^\ell} = (1 - 2^{-\ell+1} + 2^{-\ell})^{3 \cdot 2^\ell} = (1 - 2^{-\ell})^{3 \cdot 2^\ell} \leq e^{-3} < 1/4$.

For the running time analysis, we know from a famous result of Hardy and Ramanujan [HR18] that there are $\mathcal{O}(2^{\sqrt{\ell}})$ partitions of any given number ℓ. Also, it is easy to see that all the partitions of a number ℓ with size (number of parts) at most q can be generated in time $p(\ell, q)\ell^{\mathcal{O}(1)}$, where $p(\ell, q)$ is the number of such partitions. For each partition of ℓ, we get the following recurrence for the running time of the procedure Partition q-colorable.

$$T(\ell, n) \leq 3 \cdot 2^\ell \mathsf{max}\{T(\ell/2, n), 2T(2\ell/3, n)\} + n^{\mathcal{O}(1)}$$

The above running time is upper bounded by $8^\ell n^{\mathcal{O}(1)}$ in the worst case. Hence, the running time of Algorithm 2 is bounded by $\mathcal{O}(8^\ell 2^{\sqrt{\ell}} n^{\mathcal{O}(1)})$ on graphs where MAXIMUM INDEPENDENT SET can be solved in polynomial time. □

We can derandomize the procedure Partition q-colorable using (n, ℓ, ℓ') separating families instead of doing random partition many times in Steps 13 and 24. Further, we get an improved running time while derandomizing using separating families. This is because, when ℓ' varies from $\ell/2$ to $2\ell/3$, the cardinality of separating family decreases even though the running time of the recursive call in Step 26 increases.

To derandomize steps 13 and 14, we can use an (n, x, y)-separating family $\mathcal{F}$ as defined in Definition 5.5 where we put $x = \ell_i$ and $y = \ell - \ell_i$. Now, instead of getting a random partition $3 \cdot 2^\ell$ times, for each $A \in \mathcal{F}$, we put $G_1 = G[A]$ and $G_2 = G[V(G) \setminus A]$. We know that $x \geq \ell/2$, and hence the running time for the procedure Partition q-colorable in this case is bounded by the following, using the running time given by Theorem 5.6.

$$T(\ell, n) \leq 2^{o(\ell)} n \log n \max_{1/2 \leq \alpha \leq 1} \left\{ \binom{\ell}{\alpha\ell} + \binom{\ell}{\alpha\ell} T((1-\alpha)\ell, n) \right\} + n^{\mathcal{O}(1)}$$

Now, to derandomize steps 24 and 25, we look for an $(n, \ell', \ell - \ell')$-separating family $\mathcal{F}$ where $\ell/3 \leq \ell' \leq 2\ell/3$. Now, like in the earlier case, instead of getting a random partition $3 \cdot 2^\ell$ times, for each $A \in \mathcal{F}$, we put $G_1 = G[A]$ and $G_2 = G[V(G) \setminus A]$. We know that there exists an $(n, \ell', \ell - \ell')$-separating collection of size $\binom{\ell}{\ell'} 2^{o(\ell)} \log n$, which can be constructed in time $\binom{\ell}{\ell'} 2^{o(\ell)} n \log n$ [FLS14]. Hence, in this case, the running time of the procedure procedure Partition q-colorable is bounded by the following.

$$T(\ell, n) \leq 2^{o(\ell)} n \log n \max_{1/3 \leq \alpha \leq 2/3} \left\{ \binom{\ell}{\alpha\ell}, \binom{\ell}{\alpha\ell} T(\alpha\ell, n) + \binom{\ell}{(1-\alpha)\ell} T((1-\alpha)\ell, n) \right\} + n^{\mathcal{O}(1)}$$

Using the above mentioned numbers numbers, we get the following recurrence for the running time of the procedure Partition q-colorable on graph classes where MAXIMUM

Independent Set can be solved in polynomial time.

$$T(\ell, n) \leq 2^{o(\ell)} n \log n \mathsf{max}\Bigg\{ \mathsf{max}_{1/2 \leq \alpha \leq 1} \left\{ \binom{\ell}{\alpha\ell} + \binom{\ell}{\alpha\ell} T((1-\alpha)\ell, n) \right\},$$
$$\mathsf{max}_{1/3 \leq \alpha \leq 2/3} \left\{ \binom{\ell}{\alpha\ell} + \binom{\ell}{\alpha\ell} T(\alpha\ell, n) + \binom{\ell}{(1-\alpha)\ell} T((1-\alpha)\ell, n) \right\} \Bigg\} + n^{\mathcal{O}(1)}$$

The above running time is bounded by $\mathcal{O}(6.75^{\ell} 2^{o(\ell)} n^{\mathcal{O}(1)})$ in the worst case, and hence we get the following corollary.

Corollary 5.5. *The problem of finding a ℓ-sized q-colorable subgraph on perfect graphs can be solved deterministically in time $\mathcal{O}(6.75^{\ell + o(\ell)} n^{\mathcal{O}(1)})$.*

5.4 Kernelization Lower Bounds

In this section we show that Max Induced Bipartite Subgraph (i.e, q=2 in p-mcis) on perfect graphs and p-mcis on split graphs do not admit polynomial kernels unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$. In Chapter 3, we have already seen the machinery to rule out polynomial kernels.

5.4.1 Max Induced Bipartite Subgraph on Perfect Graphs

The Max Induced Bipartite Subgraph problem is formally given as follows:

p-Max Induced Bipartite Subgraph (p-mibs)	**Parameter(s):** k
Input: A graph G and a positive integer k.	
Question: Does there exist $S \subseteq V$ such that $\lvert S \rvert = k$ and $G[S]$ is a bipartite graph?	

Here, we show that unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$, p-mibs does not have a polynomial kernel when restricted to perfect graphs. We note that we are dealing here with the case of finding a maximum induced bipartite subgraph in the interest of exposition; a more general result that shows the hardness of finding a maximum induced q-colorable subgraph for any fixed $q \geq 2$ on the class of perfect graphs is described in the next subsection.

Our result here is established by demonstrating an OR-composition to p-MIBS from unparameterized version of the same problem. We define the polynomial equivalence relation to be such that $(G_1\#1^{k_1})$ and $(G_2\#1^{k_2})$ are in the same equivalence class if and only if $k_1 = k_2$. Let $(G_0\#1^k), (G_1\#1^k), \ldots, (G_{t-1}\#1^k)$ be t instances of p-MIBS from the same equivalence class of the polynomial equivalence relation, where every G_i is a perfect graph. In slight abuse of notation, we will be referring to $(G_i\#1^k)$ as (G_i, k) for the sake of simplicity.

We assume that $t = 2^z$ (note that we assume equality without loss of generality, since whenever t is not a power of 2, the set of instances can be padded with trivial No instances). We construct a composed instance (G, k^*) as follows. To begin with, let G be the disjoint union of all G_i, $0 \leq i \leq t-1$. For all $i \neq j$ add all possible edges between G_i and G_j. Now add $2k \log t$ identity gadgets, named H_{ij} for $1 \leq i \leq 2k$, $1 \leq j \leq \log t$. The gadget H_{ij} consists of eight vertices $\{x_{ij}, y_{ij}, w_{ij}, z_{ij}, a_{ij}, b_{ij}, c_{ij}, d_{ij}\}$, where the vertices $\{x_{ij}, y_{ij}, w_{ij}, z_{ij}\}$ form a clique, and the vertex a_{ij} is adjacent to x_{ij} and w_{ij}; b_{ij} is adjacent to x_{ij} and z_{ij}; c_{ij} is adjacent to w_{ij} and y_{ij} and d_{ij} is adjacent to y_{ij} and z_{ij} (see Fig. 5.1).

For all $0 \leq l \leq t-1$, if the j^{th} bit of the $\log t$-bit binary representation of l is 0, then add edges from all vertices in G_l to x_{ij} and y_{ij}. Otherwise add edges from all vertices in G_l to w_{ij} and z_{ij}. This completes the description of the composed graph; we let $k^* = k + 12k \log t$. Hence, we have shown that $k^* = t^{o(1)}(\mathsf{max}_i |x_i|)^c$, since $|x_i| \geq k$ for all $i \in \{0, \ldots, t-1\}$. We first show that this is indeed a valid OR-composition, and then demonstrate that G, as described, is a perfect graph.

Lemma 5.6. *The instance $(G, k + 12k \log t)$ is a* YES *instance of p-*MIBS *if, and only if, (G_ℓ, k) is a* YES *instance of p-*MIBS *for some $0 \leq \ell \leq (t-1)$.*

Proof. ($\Rightarrow$) Assume $(G, k + 12k \log t)$ is a YES instance of p-MIBS and let $S \subseteq V(G)$ be a solution. We first claim that S will not contain vertices from more than two input instances. Let us assume for the sake of contradiction that this is not the case. Then for $i_1 \neq i_2 \neq i_3$, let $v_{i_1} \in S \cap V(G_{i_1})$, $v_{i_2} \in S \cap V(G_{i_3})$ and $v_{i_3} \in S \cap V(G_{i_3})$. Note that $v_{i_1}, v_{i_2}, v_{i_3}$ will

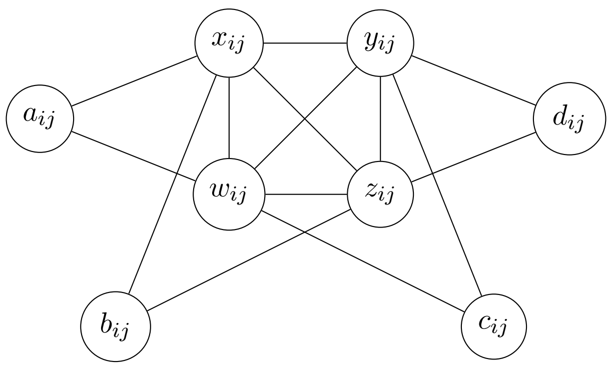


Figure 5.1: Identity gadget H_{ij}

induce a triangle and contradict the fact that $G[S]$ is bipartite. So now we can assume that S contains vertices from two input graphs G_p and G_q. If one of them has at least k vertices in S, then we are done. Otherwise, $|S \cap V(G_p)| + |S \cap V(G_q)| < 2k$. Hence,

$$\sum_{i=1}^{2k} \sum_{j=1}^{\log t} |S \cap V(H_{ij})| > k + 12k \log t - 2k \geq 12k \log t - k \tag{5.1}$$

Therefore there exists an i' such that $\sum_{j=1}^{\log t} |S \cap V(H_{i'j})| \geq 6 \log t$. Since vertices $x_{ij}, y_{ij}, w_{ij}, z_{ij}$ from H_{ij} form a complete graph, S can contain at most 2 vertices from $\{x_{ij}, y_{ij}, w_{ij}, z_{ij}\}$. So $|S \cap V(H_{ij})| \leq 6$ and if $|S \cap V(H_{ij})| = 6$ then either $S \cap V(H_{ij}) = \{a_{ij}, b_{ij}, c_{ij}, d_{ij}, x_{ij}, y_{ij}\}$ or $S \cap V(H_{ij}) = \{a_{ij}, b_{ij}, c_{ij}, d_{ij}, w_{ij}, z_{ij}\}$. We know that to meet the budget, it must be the case that for all j, $|S \cap V(H_{i'j})| = 6$.

Since $p \neq q$ there exists a j' such that j'^{th} bit of binary representation of p and q are different (say 0 and 1, respectively). Hence, all the vertices from G_p are connected to $x_{i'j'}, y_{i'j'}$ and all the vertices from G_q are connected to $w_{i'j'}, z_{i'j'}$. Hence there exists a triangle in $G[S \cap (V(G_p) \cup V(G_q) \cup V(H_{i'j'}))]$. This contradicts the fact that $G[S]$ is bipartite, showing that the case $|S \cap V(G_p)| + |S \cap V(G_q)| < 2k$ is infeasible. The remaining case is when S contains vertices from at most one input graph (say G_p). Since $|S \cap V(H_{ij})| \leq 6$, S will contain at least k vertices from $V(G_p)$. Hence $S \cap V(G_p)$ is a solution of (G_p, k).

($\Leftarrow$) Let (G_p, k) be a Yes instance of p-mibs, and let $S \subseteq V(G_p)$ be the solution. Let

$b_1 b_2 \dots b_{\log t}$ be the binary representation of p. Now consider the vertex set

$$\begin{aligned} T := \ & \{x_{ij}, y_{i,j} \mid 1 \leq i \leq 2k \ \wedge \ b_j = 1\} \cup \{w_{ij}, z_{i,j} \mid 1 \leq i \leq 2k \ \wedge \ b_j = 0\} \\ & \cup \{a_{ij}, b_{ij}, c_{ij}, d_{ij} | 1 \leq i \leq 2k \ \wedge \ 1 \leq j \leq \log t\}. \end{aligned} \tag{5.2}$$

It is easy to see that T involves exactly six vertices from each of the $2k \log t$ gadgets, and the vertices are chosen such that $G[T]$ induces a bipartite graph. Further, the vertices are chosen to ensure that there are no edges between vertices in S and vertices in T, and therefore, it is clear that $G[S \cup T]$ induces a bipartite subgraph of G of the desired size. Hence $(G, k + 12k \log t)$ is a YES instance of p-MIBS. □

Lemma 5.7. *The graph G constructed as the output of the* OR*-composition is a perfect graph.*

Proof. We begin by describing an auxiliary graph G', and show that G' is perfect. This graph is designed to be a graph from which G can be obtained by a series of operations that preserve perfectness, and this will lead us to establishing that G is perfect. The graph G' contains a clique on t vertices, K_t. We let $V(K_t) := \{v_0, v_1, \dots v_{t-1}\}$. G' also contains $2k \log t$ small graphs, each of which consist of two vertices with an edge between them (i.e, each small graph is an edge). Let $\{n_{ij}, p_{ij}\}$ for all $1 \leq i \leq 2k$, $1 \leq j \leq \log t$ be the vertices of small graphs. For all $0 \leq l \leq t-1$, if the j^{th} bit of the $\log t$-bit binary representation of l is 0, then add edges from v_l to n_{ij} for all i. Otherwise add edges from v_l to p_{ij} for all i.

We claim the G' is perfect. Let H be an induced subgraph of G'. If $|V(H) \cap V(K_t)| \leq 1$, then H is a forest and so in this case $\omega(H) = \chi(H)$. Otherwise $r = |V(H) \cap V(K_t)| \geq 2$. Since the neighborhoods of n_{ij} and p_{ij} do not intersect, and there are no edges between small graphs in G', at most one vertex from the entire set of small graphs can be part of the largest clique in H containing $V(H) \cap V(K_t)$ (note that there exists a largest clique that contains all the vertices in $V(H) \cap V(K_t)$). So $\omega(H) \leq r + 1$. Let us denote by H^* the subgraph $H[V(H) \cap \{n_{ij}, p_{ij} \mid 1 \leq i \leq 2k, 1 \leq j \leq \log t\}]$.

If $\omega(H) = r + 1$, then we define the following coloring. Color all r vertices in $V(H) \cap V(K_t)$ with colors $1, 2, \dots, r$. For all $x \in V(H^*)$ such that x is adjacent to all vertices in

$V(H) \cap V(K_t)$, we give a color $r+1$ (note that these vertices are independent by construction). If an $x \in V(H^*)$ is not adjacent to all vertices in $V(H) \cap V(K_t)$, then we can color it with a color that is already used on one of its non adjacent vertices in $V(H) \cap V(K_t)$. If $\omega(H) = r$, then there is no vertex in $V(H^*)$ which is adjacent to $V(H) \cap V(K_t)$. So we can color vertices in $V(H) \cap V(K_t)$ with r colors and for a vertex $x \in V(H^*)$ we can color x with a color same as (one of) its non adjacent vertex in $V(H) \cap V(K_t)$.Hence $\omega(H) = \chi(H)$.

Let G^* be a graph obtained by embedding G_i on $v_i \in V(G')$ for all $0 \leq i \leq t-1$ and embedding an edge on each vertex in $\{n_{ij}, p_{ij} \mid 1 \leq i \leq 2k, 1 \leq j \leq \log t\}$. It can be observed that G^* is isomorphic to

$$G \setminus \bigcup_{1 \leq i \leq 2k, 1 \leq j \leq \log t} \{a_{ij}, b_{ij}, c_{ij}, d_{ij}\}.$$

It follows that G^* is perfect. Finally, observe that the graph G is $Triangular(G^*;\ E')$ for a suitable choice of $E' \subseteq E(G^*)$, and it follows that G is perfect. □

Lemmas 5.6, 5.7 and Theorem 3.4 give us the following result.

Theorem 5.8. *p-*Max Induced Bipartite Subgraph *on perfect graphs does not admit a polynomial kernel unless* NP $\subseteq$ coNP/poly*.*

5.4.2 p-q-MCIS on Perfect Graphs

We now show the hardness of finding a maximum induced q-colorable subgraph for any fixed $q \geq 2$ on the class of perfect graphs.

Theorem 5.9. *p-q-*MCIS *for a fixed q on perfect graphs does not admit polynomial kernel unless* NP $\subseteq$ coNP/poly*.*

Proof. We prove the theorem using OR-composition. As in the case of p-MIBS, we show an OR-composition from the unparameterized version of the same problem and denote the instances of it by (G_i, k). Also, we define the polynomial equivalence relation to be such that two instances (G_1, k_1) and (G_2, k_2) lie in the same equivalence class if and only if $k_1 = k_2$. Let the input instances of the same equivalence class for the OR-composition algorithm be $(G_0, k), (G_1, k), \ldots, (G_{t-1}, k)$. Now we construct an instance (G, k) as follows. For all $i \neq j$ add all possible edges between G_i and G_j. Now add $2qk \log t$ identity gadgets, named H_{ij} for $1 \leq i \leq 2qk,\ 1 \leq j \leq \log t$, as follows. Each H_{ij} contain two cliques K^0_{ij} and K^1_{ij} of size q each. We add all possible edges between K^0_{ij} and K^1_{ij}. Let

$$I_{ij} = \{(C_0, C_1) \mid C_0 \subseteq V(K^0_{ij}), C_1 \subseteq V(K^1_{ij}), |C_0| + |C_1| = q, 0 < |C_0|, |C_1| < q\}$$

Now for each $(X, Y) \in I_{ij}$ we add a vertex $v_{X,Y}$ to H_{ij} and add edges $\{(v_{X,Y}, u) \mid (X, Y) \in I_{ij}, u \in X \vee u \in Y\}$. Let $V_{ij} = \{v_{X,Y} \mid (X, Y) \in I_{ij}\}$ and $p = |I_{ij}| = |V_{ij}|$. Note that p is a function of q only. For all $0 \leq \ell \leq t-1$, if the j^{th} bit of the $\log t$-bit binary representation of ℓ is 0, then add edges from all vertices in G_ℓ to all vertices in K^0_{ij} for all i. Otherwise add edges from all vertices in G_l to all vertices in K^1_{ij} for all i. The graph G, so far constructed along with parameter $k + 2qk(p + q) \log t$ is the output of the OR-composition algorithm.

Now we show that $(G, k + 2qk(p + q) \log t)$ is a YES instance of p-MCIS if and only if there exists ℓ such that (G_ℓ, k) is a YES instance of p-MCIS.
($\Leftarrow$) Let (G_l, k) be a YES instance of p-MCIS. Let $S \subseteq V(G_\ell)$ be the solution. Let $b_1 b_2 \ldots b_{\log t}$ be the binary representation of ℓ. Now consider the vertex set

$$T = \bigcup_{ij} \left(V\left(K^{1-b_j}_{ij}\right) \cup V_{ij} \right).$$

It is easy to see that $|T| = 2qk(p + q) \log t$. We claim that $G[T]$ is q-colorable. For that it is enough to show that $G[T \cap V(H_{ij})]$ is q-colorable because there are no edges between identity gadgets. Consider $G[T \cap V(H_{ij})]$ for any fixed i, j. Let $\{k_1, k_2, \ldots, k_q\} = V(K^{1-b_j}_{ij})$. We keep each k_r in color class r. Since for each $v_{X,Y} \in V_{ij}$, there exists $k_s \in V(K^{1-b_j}_{ij})$ such that $(k_s, v_{X,Y}) \notin E(G)$, we can keep $v_{X,Y}$ in color class s. Also note that V_{ij} form an independent set. Hence $G[T \cap V(H_{ij})]$ is q-colorable. Since there is no edges between S and T, $G[S \cup T]$ is a q-colorable induced subgraph of G, of size $k + 2qk(p + q) \log t$.
($\Rightarrow$) Assume $(G, k + 2qk(p + q) \log t)$ is a YES instance of q-MCIS. Let $S \subseteq V(G)$ be the solution set. We claim S will not contain vertices from more than q input instances. Suppose not, then S will contain a $q + 1$ clique, which contradicts the fact that $G[S]$ is

q-colorable. So assume that S contain vertices from at most q input graphs $G_{i_1}, \ldots, G_{i_h}$ where $h \leq q$. If one of them has at least k vertices in S, then we are done. Otherwise $\sum_{j=1}^{h} |S \cap V(G_{i_j})| < qk$. Hence

$$\begin{aligned} \sum_{i,j} |S \cap V(H_{ij})| &\geq k + 2qk(p+q)\log t - qk && (5.3) \\ &\geq 2qk(p+q)\log t - (q-1)k && (5.4) \end{aligned}$$

Therefore, there exists i' such that $\sum_{j=1}^{\log t} |S \cap V(H_{i'j})| \geq (p+q)\log t$. Since $G[V(K_{ij}^0) \cup V(K_{ij}^1)]$ is a complete graph, S can contain at most q vertices from $V(K_{ij}^0) \cup V(K_{ij}^1)$. So $|S \cap V(H_{ij})| \leq (p+q)$. If $|S \cap V(H_{ij})| = (p+q)$ then either $S \cap V(H_{ij}) = V(K_{ij}^0) \cup V_{ij}$ or $S \cap V(H_{ij}) = V(K_{ij}^1) \cup V_{ij}$ because if S contain $q_1 (> 0)$ vertices from $V(K_{ij}^0)$ and $q_2 (> 0)$ vertices from $V(K_{ij}^1)$, then there exist a vertex in V_{ij} that can not be part of S. Hence for all j, $|S \cap V(H_{i'j})| = (p+q)$ and for all j, $V(K_{ij}^0) \subseteq S \cap V(H_{i'j})$ or $V(K_{ij}^1) \subseteq S \cap V(H_{i'j})$, but not both. Since $i_1 \neq i_2$ there exists a j' such that j'^{th} bit of binary representation of i_1 and i_2 are different (say it is 0 and 1 resp.). Hence all the vertices from G_{i_1} is connected to $V(K_{ij}^0)$ and all the vertices from G_{i_2} is connected to $V(K_{ij}^0)$. Hence there exist a $q+1$ sized clique in $G[S \cap (V(G_{i_1}) \cup V(G_{i_2}) \cup V(H_{i'j'}))]$. It contradicts the fact that $G[S]$ is q-colorable. Therefore S contain vertices from one input graph (say G_ℓ) only and since $|S \cap V(H_{ij})| \leq (p+q)$, S will contain at least k vertices from $V(G_\ell)$. Hence $S \cap V(G_l)$ is a solution of (G_ℓ, k).

Now we show that G is a perfect graph. Consider the following graph G'. G' contains a clique on t vertices, K_t. Let the vertices of K_t are named $v_0, v_1, \ldots v_{t-1}$. G' also contains $2qk \log t$ small graphs, on two vertices and one edge each (i.e each small graph is an edge). Let n_{ij}, p_{ij} for all $1 \leq i \leq 2qk$, $1 \leq j \leq \log t$ be the vertices of small graphs. For all $0 \leq \ell \leq t-1$, if the j^{th} bit of the $\log t$-bit binary representation of ℓ is 0, then add edges from v_l to n_{ij} for all i. Otherwise add edges from v_l to p_{ij} for all i. Using similar arguments in the proof of lemma 5.7 we can show that G' is a perfect graph. Let be G'' be a graph obtained by embedding G_i on $v_i \in V(G')$ for all $0 \leq i \leq t-1$ and embedding a clique of size q on each vertex in $\{n_{ij}, p_{ij} \ : \text{ for all } i, j\}$. So G'' is a perfect graph. It can be observed that G'' is isomorphic to $G \setminus \bigcup_{ij} V_{ij}$. We claim that if $G'' = (V, E)$ is perfect graph and

$X \subseteq V$ such that $G[X]$ is a clique, then the graph $G^* = (V \cup \{u\}, E \cup \{(u,x) \mid x \in X\})$ is a perfect graph. Let H be an induced subgraph of G^*. If $u \notin V(H)$, then $w(H) = \chi(H)$ because H is an induced subgraph of a perfect graph G''. Now consider the case $u \in V(H)$. We know that $w(H \setminus \{u\}) = \chi(H \setminus \{u\})$. Let $d = w(H \setminus \{u\}) = \chi(H \setminus \{u\})$. Since $G[N_H(u)]$ is a clique, $d \geq N_H(u)$. If $d > N_H(u)$, the largest clique size in H will be d and we can color H with d colors by giving color to u which is not the color of any of its neighbors. So $w(H) = \chi(H)$. If $d = N_H(u)$, the largest clique size in H is $d+1$ ($G[u \cup N_H(u)]$) and we can color H using $d+1$ colors by giving a new color to u. Hence G^* is a perfect graph. Note that we can get the graph G (we constructed for OR-composition) by repeatedly applying the above operation on G'' using vertices from $\bigcup_{ij} V_{ij}$. Therefore G is a perfect graph. □

5.4.3 p-MCIS on Split Graphs

We now show that p-MCIS does not admit polynomial kernel unless NP $\subseteq$ coNP/poly by showing a PPT reduction from Small Universe Set Cover.

Small Universe Set Cover	**Parameter(s):** n
Input: A set $U = \{u_1, \ldots, u_n\}$, a family $\mathcal{F}$ of subsets of X and an integer k.	
Question: Does there exist a subfamily $\mathcal{F}' \in \binom{\mathcal{F}}{k}$ such that $\bigcup_{S \in \mathcal{F}'} S = U$?	

We have the following theorem due to Dom et al [DLS09].

Theorem 5.10 ([DLS09])**.** Small Universe Set Cover *does not admit polynomial kernel unless* NP $\subseteq$ coNP/poly.

In fact Dom et al [DLS09] showed that Small Universe Set Cover parameterized by n and k does not admit polynomial kernel unless NP $\subseteq$ coNP/poly. Since $k \leq n$ for all non-trivial cases, we have Theorem 5.10.

Lemma 5.8. *There is a polynomial parameter transformation from* Small Universe Set Cover *to p-*MCIS *on split graphs.*

Proof. The reduction we give here is along the lines of the NP-completeness reduction for

p-MCIS by Yannakakis and Gavril [YG87]. Given an instance $(U, \mathcal{F}, k)$ of SMALL UNIVERSE SET COVER, we construct an instance (G, ℓ, q) of p-MCIS as follows. The split graph G has vertex set $(X \cup \mathcal{F})$ with X being the independent set, and $\mathcal{F}$ inducing the clique. For any $u \in X$ and $S \in \mathcal{F}$ we add an edge (u, S) if and only if $u \notin S$. We set $\ell = n + k$ and $q = k$. Since $k \leq n$, $\ell \leq 2n$.

We claim that $(U, \mathcal{F}, k)$ is a YES instance of SMALL UNIVERSE SET COVER if and only if (G, ℓ, q) is a YES instance of p-MCIS. Suppose $(U, \mathcal{F}, k)$ is a YES instance of SMALL UNIVERSE SET COVER and let $S_1, S_2, \ldots, S_k$ be a solution. The graph induced on $X \cup \{S_1, S_2, \ldots, S_k\}$ is k colorable because the vertex S_i with its non-neighbors in X (they are exactly the elements in the set S_i) form an independent set. Suppose, on the other hand, that (G, ℓ, q) is a YES instance of p-MCIS. Let H be a q-colorable subgraph of G. Since vertices in $\mathcal{F}$ form a clique, $|V(H) \cap \mathcal{F}| = k$ and so let $\{S_1, \ldots, S_k\} = V(H) \cap \mathcal{F}$. Hence $X \subseteq V(H)$. Let $V_1, \ldots, V_k$ be the q color classes in H. Since $S_1, \ldots, S_k$ form a clique, for all $i \neq j$, S_i and S_j will be in two different color classes. Now it is easy to see that corresponding sets $S_1, \ldots, S_k$ cover U, because for each $u \in U$, u is covered by S_i where $u, S_i \in V_j$ for some j. $\square$

Using Theorem 3.6, Theorem 5.10 and Lemma 5.8, we get the following

Theorem 5.11. *p-MCIS on split graphs does not admit a polynomial kernel unless* NP $\subseteq$ coNP/poly.

5.5 Conclusion

In this chapter we studied the parameterized complexity of p-MCIS on perfect graphs and showed that the problem is FPT when parameterized by the solution size. We also studied its kernelization complexity and showed that the problem does not admit polynomial kernel under certain complexity theory assumptions. An interesting direction of research that this chapter opens up is the study of parameterized complexity of INDUCED SUBGRAPH ISOMORPHISM on special graph classes. As a first step it would be interesting to study the parameterized complexity of INDUCED TREE ISOMORPHISM parameterized by the size of the tree on perfect graphs.

Part III

Algorithms for Graph Modification Problems

Chapter 6

Parameterized Algorithms for deletion to Split Graphs

6.1 Introduction

The problem of editing (adding/deleting vertices/edges) to ensure that a graph has some property is a well studied problem in theory and applications of graph algorithms. When we want the resulting graph to be in a non-trivial hereditary graph class Π, the optimization versions of the corresponding vertex deletion problems are known to be NP–complete by a classical result of Lewis and Yannakakis [LY80]. Many edge deletion problems (including deletion to split graphs) are known to be NP–complete by results of Natanzon et al. [NSS01]. This problem has also been studied in generality under paradigms like approximation [Fuj98, LY94] and parameterized complexity [Cai96, Guo07]. When Π is a specific hereditary class like chordal or planar graphs, extensive work has been done to explore tight bounds [FV12, MS12, Mar10, HvtHJ$^+$11].

In this chapter, we do a study of these problems from the point of view of parameterized complexity when Π is the class of all split graphs, which is also a non-trivial hereditary graph class. An undirected graph $G = (V, E)$ is said to be *split* if its vertex set V can be partitioned into two sets such that the induced subgraph on one of them is a complete graph and the induced subgraph on the other is an independent set. Split graphs were first studied by Földes and Hammer [FH77], and independently introduced by Tyshkevich and Chernyak [TC90]. In [FH77], the authors provided the following finite forbidden subgraph characterization of split graphs which gives an easy polynomial time algorithm for recognizing split graphs.

Lemma 6.1. ([FH77]) *A graph is a split graph if and only if it contains no induced subgraph isomorphic to $2K_2$, C_4, or C_5. Here, K_2 is the complete graph on two vertices, C_i is a cycle on i vertices.*

In this chapter, we study the following two problems.

Split Vertex Deletion (SVD) **Parameter(s):** k

Input: Graph $G = (V, E)$, integer k

Question: Does there exist a set of vertices of size at most k whose deletion from G results in a split graph?

Split Edge Deletion (SED) **Parameter(s):** k

Input: Graph $G = (V, E)$, integer k

Question: Does there exist a set of edges of size at most k whose deletion from G results in a split graph?

As the size of the forbidden set is finite, these problems become fixed-parameter tractable (see Section 6.2) due to a general result of Cai [Cai96], when parameterized by k. One can also observe from Lemma 6.1, a fairly straightforward branching algorithm for both SVD and SED with running time $\mathcal{O}^*(5^k)$.

In [LNR+14], the authors obtained an $\mathcal{O}^*(2.32)^k$ algorithm for SVD by reducing the problem to the Above Guarantee Vertex Cover problem and using the fixed-parameter algorithm for it. In this chapter, we obtain an $\mathcal{O}^*(2^k)$ algorithm by the combination of a bound on the number of split partitions of a split graph, and the well known technique of iterative compression. We also obtain an $\mathcal{O}(k^3)$ vertex kernel for the problem. Note that, this kernel is smaller than the kernel with $\mathcal{O}(k^4)$ vertices, which can be obtained by an approach similar to d-Hitting Set [AK10]. We also prove that under certain complexity theoretic assumptions, we cannot obtain a subexponential algorithm for this problem.

For SED, we design a subexponential algorithm running in time $\mathcal{O}^*(2^{\mathcal{O}(\sqrt{k}\log k)})$ by combining the color and conquer approach [ALS09], with the bound on the number of partitions of a split graph. This is one of the very few problems (see [FV12, FKP$^+$11] for other problems) for which we know a subexponential parameterized algorithm on general graphs which does not use bidimensionality theory. We also revisit the kernelization algorithm for this problem given by Guo [Guo07], and by using only a subset of the rules presented there, we prove a bound of $\mathcal{O}(k^2)$ vertices improving on Guo's bound of $\mathcal{O}(k^4)$. Furthermore, the Split Completion problem of adding at most k edges to a given graph to make it split, is equivalent to deleting at most k edges from the complement of the graph to make it split. Hence, the bound on the kernel and the subexponential algorithm which we prove for SED also holds for Split Completion.

Related Work. We also note that though computing a minimum split completion set is NP-complete, there is a linear time algorithm to compute a *minimal* split completion set [HM09]. Also, our algorithm for SVD was improved by Cygan and Pilipczuk [CP13] using a different method. Their algorithm runs in time $\mathcal{O}^*(1.2738^k k^{\mathcal{O}(\log k)})$.

Organization of the chapter. In Section 6.3, we first present and analyze our algorithm for Split Vertex Deletion. Following that, we design a set of reduction rules for the problem which allow us to reduce the number of vertices in the graph to $\mathcal{O}(k^3)$. In Section 6.4, we present the subexponential algorithm for Split Edge Deletion, which is followed by an improved bound on a kernel for the same problem.

6.2 Preliminaries

Given a function $col : V \to C$ from the vertices of the graph G to a set of colors, C, we say that an edge $(u, v) \in E$ is monochromatic if $col(u) = col(v)$ and non-monochromatic otherwise.

A graph G is called a *split graph* if the vertex set V can be partitioned into two sets V_1 and V_2 such that $G[V_1]$ is a complete graph and $G[V_2]$ is an independent set. We call a set $S \subseteq V$ a *split vertex deletion* (svd) set if the graph $G[V \setminus S]$ is a split graph and a set $A \subseteq E$ is called a *split edge deletion* (sed) set if the graph $G[E \setminus A]$ is a split graph.

Definition 6.1. *Given a split graph $G = (V, E)$, a partition $(C \uplus I)$ of the vertex set into sets C and I is called a* split partition *of this split graph if $G[C]$ is a clique and $G[I]$ is an independent set.*

Given a split partition $(C_0 \uplus I_0)$ of a subgraph G' of a split graph G, we say that a split partition $(C \uplus I)$ of G is *consistent* with the partition $(C_0 \uplus I_0)$ if $C_0 \subseteq C$ and $I_0 \subseteq I$. We refer to an induced subgraph isomorphic to $2K_2$, or C_4 or C_5 as a *forbidden structure.*

6.3 Split Vertex Deletion

In this section, we first present an $\mathcal{O}^*(2^k)$ parameterized algorithm for SVD by combining the technique of iterative compression along with a linear bound on the number of split partitions of split graphs. This bound has been improved by Cygan and Pilipczuk [CP13], but we give this algorithm for completeness. Later, we give a cubic kernel for SVD.

6.3.1 An $\mathcal{O}^*(2^k)$ algorithm for Split Vertex Deletion

We start by stating a lemma, that is implied by Theorem 6.2, [Gol80].

Lemma 6.2. (Theorem 6.2, [Gol80]) *A split graph on n vertices can have at most $n + 1$ split partitions.*

We will now describe the application of the iterative compression technique to the SVD problem.

Iterative Compression for Split Vertex Deletion. Given an instance $(G = (V, E), k)$ of SVD, we let $V = \{v_1, \ldots, v_n\}$ and define vertex sets $V_i = \{v_1, \ldots, v_i\}$, and let the graph $G_i = G[V_i]$. We iterate through the instances (G_i, k) starting from $i = k + 3$. For the i^{th} instance, we try to find a solution $\hat{S}_i$ of size at most k, with the help of a *known* solution S_i of size at most $k + 1$. Formally, the compression problem we address is the following.

Split Vertex Deletion Compression (SVD Compression) **Parameter(s):** k

Input: Graph $G = (V, E)$, an svd set $S \subseteq V$ of size at most $k + 1$, integer k

Question: Does there exist an svd set of size at most k?

We reduce the SVD problem to $n - k - 2$ instances of the SVD Compression problem as follows. Let $I_i = (G_i, S_i, k)$ be the i^{th} instance. Clearly, the set V_{k+1} is a solution of size at most $k + 1$ for the instance I_{k+3}. It is also easy to see that if $\hat{S}_{i-1}$ is a solution of size at most k for instance I_{i-1}, then the set $\hat{S}_{i-1} \cup \{v_i\}$ is a solution of size at most $k + 1$ for the instance I_i. We use these two observations to start off the iteration with the instance $(G_{k+3}, S_{k+3} = V_{k+1}, k)$ and look for a solution of size at most k for this instance. If there is such a solution $\hat{S}_{k+3}$, we set $S_{k+4} = \hat{S}_{k+3} \cup \{v_{k+4}\}$ and ask for a solution of size at most k for the instance I_{k+4} and so on. If, during any iteration, the corresponding instance does not have a solution of the required size, it implies that the original instance is also a No instance. This follows from the fact that if a graph G has a split vertex deletion set of size k, then any vertex induced subgraph of G also has a split vertex deletion set of size k. Finally, the solution for the original input instance will be $\hat{S}_n$. Since there can be at most n iterations, the total time taken to solve the original instance is bounded by n times the time required to solve the SVD Compression problem.

Our algorithm for SVD Compression is as follows. Let the input instance be $I = (G = (V, E), S, k)$. We guess a subset $Y \subseteq S$ with the intention of picking these vertices in our hypothetical solution for this instance and ignoring the rest of the vertices in S. We delete the set Y from the graph and decrease k appropriately. We then check if the graph $G[S \setminus Y]$ is a split graph and if it is not, then reject this guess of Y as a spurious guess. Suppose that $G[S \setminus Y]$ is indeed a split graph. We now guess and fix a split partition $(C_0 \uplus I_0)$ for this graph. By Lemma 6.2, we know that there are at most $k + 2$ such split partitions. The split partition we fix corresponds to the split partition *induced* by the hypothetical solution on the graph $G[S \setminus Y]$. Hence, it now remains to check if there is an SVD set of the appropriate size which is disjoint from $S \setminus Y$, and results in a split graph with a split partition *consistent* with $(C_0 \uplus I_0)$. More formally, we have an instance of the following problem.

SVD COMPRESSION* **Parameter(s):** k

Input: Graph $G = (V, E)$, an svd set $S \subset V$ such that $G[S]$ is a split graph, a split partition $(C_0 \uplus I_0)$ for the graph $G[S]$, integer k

Question: Does there exist an svd set X of size at most k, disjoint from S such that $G \setminus X$ has a split partition consistent with $(C_0 \uplus I_0)$?

The following lemma gives a polynomial time algorithm for the above problem.

Lemma 6.3. SVD COMPRESSION* *can be solved in $\mathcal{O}(n^3)$ time.*

Proof. Let S' be a potential solution, and let $(C' \uplus I')$ be a fixed split partition for the graph $G \setminus S'$ consistent with the split partition $(C_0 \uplus I_0)$. Let $(C_1 \uplus I_1)$ be a split partition of the graph $G \setminus S$.

Since we cannot delete edges, at most one vertex of C_1 can lie in I' and at most one vertex of I_1 can lie in C'. Hence, we initially guess these two vertices (either guess could be empty) v_c and v_i where $v_c = C_1 \cap I'$ and $v_i = I_1 \cap C'$. We move v_c to I_1 and v_i to C_1. For the sake of convenience we refer to the modified sets C_1 and I_1 also as C_1 and I_1. Now, let $\hat{I} = I_0 \cup I_1$ and $\hat{C} = C_0 \cup C_1$. It is clear that any vertex in $\hat{I}$ which is neighbor to a vertex in $I_0 \cup \{v_c\}$ needs to be deleted and any vertex in $\hat{C}$ which is not global to $C_0 \cup \{v_i\}$ needs to be deleted. Let X be the set of these vertices, that need to be deleted. Now, if X is not disjoint from S, we return NO. On the other hand, we observe that if X is disjoint from S, then deleting X gives us the required kind of split graph. To show that, we look at the partition $(\hat{C} \setminus X) \uplus (\hat{I} \setminus X)$ of $G \setminus X$. The set $\hat{C} \setminus X$ is a clique because $C_0 \cup \{v_i\}$ is a clique (otherwise we would have returned NO earlier), $C_1 \setminus X$ is a clique, and all the edges between $C_0 \cup \{v_i\}$ and $C_1 \setminus X$ are present. Similarly, the set $\hat{I} \setminus X$ is an independent set because $I_0 \cup \{v_c\}$ is an independent set (otherwise we would have returned NO earlier), $I_1 \setminus X$ is an independent set, and no edges between $I_0 \cup \{v_c\}$ and $I_1 \setminus X$ are present. Hence, if $|X| \leq k$, then we return that it is indeed a YES instance and return NO otherwise. Guessing the vertices takes $\mathcal{O}(n^2)$ time and in each iteration we spend at most linear time, hence the algorithm takes $\mathcal{O}(n^3)$ time. ☐

Given Lemma 6.3, our algorithm for SVD COMPRESSION has a running time of $\mathcal{O}(\Sigma_{i=0}^{k}\binom{k+1}{i} \cdot k \cdot n^{\mathcal{O}(1)}) = \mathcal{O}^*(2^k)$, where the factor of k is due to the number of split partitions of $G[S \setminus Y]$ and $n^{\mathcal{O}(1)}$ is due to the time required to execute our algorithm for SVD COMPRESSION*.

Finally, since we solve at most n instances of SVD COMPRESSION, our algorithm for SVD runs in time $\mathcal{O}^*(2^k)$, giving us the following theorem.

Theorem 6.2. SPLIT VERTEX DELETION *can be solved in* $\mathcal{O}^*(2^k)$ *time.*

We now show that with respect to the asymptotic dependence on k, the above algorithm for SPLIT VERTEX DELETION is essentially the best we can hope for.

Theorem 6.3. SPLIT VERTEX DELETION *cannot be solved in time* $\mathcal{O}^*(2^{o(k)})$ *time unless ETH fails.*

Proof. It is known that VERTEX COVER does not admit a subexponential algorithm unless the Exponential Time Hypothesis (ETH) fails [CJ03]. We prove the analogous statement for SPLIT VERTEX DELETION by a reduction from VERTEX COVER.

Consider an instane (G, k) of VERTEX COVER and let G' be the graph constructed from G by adding a disjoint clique of size $k + 2$. Now, G has a vertex cover of size at most k if and only if G' has a svd set of size at most k. This concludes the proof. ☐

6.3.2 A cubic kernel for SPLIT VERTEX DELETION

In this subsection, we use the structural claim made in the algorithm for SVD to design a vertex kernel of size $\mathcal{O}(k^3)$ for SVD. We design the kernel by introducing reduction rules which can be applied in polynomial time to reduce the instance. The reduction rules we present here are applied exhaustively and in the order in which they are presented.

We say that a reduction rule that is applied on an instance (G, k) to produce an instance (G', k') is *correct* if (G, k) is a YES instance if and only if (G', k') is a YES instance.

Reduction Rule 1. *Compute an inclusion wise maximal set of vertex disjoint forbidden structures greedily, and let the set of vertices involved in this set of forbidden structures be D. If $|D|$ exceeds $5k$, then return a trivial* NO *instance.*

Moving forward, we assume that $|D| \leq 5k$. Note that $G \setminus D$ is a split graph, and let $(C^* \uplus I^*)$ be a split partition of this graph. Before we present the next reduction rule, we need the following definition.

Definition 6.4. *We say that a vertex v of G has a* high clique non-neighborhood *if $|C^* \setminus N(v)| \geq k + 2$. Similarly, v is said to have a* high independent set neighborhood *if $|I^* \cap N(v)| \geq k + 2$.*

Let $H_i = \{x \in V : |C^* \setminus N(v)| \geq k + 2\}$, and let $H_c = \{x \in V : |I^* \cap N(v)| \geq k + 2\}$.

Lemma 6.4. *The vertices in H_i will either end up in the independent partition of the resulting split graph, or will get deleted and hence will be in the solution. Similarly for vertices in H_c, will either end up in the clique partition of the resulting split graph, or will get deleted and hence will be in the solution.*

Proof. The vertices in H_i have at least $k + 2$ non-neighbors in the clique partition. So, if a vertex of H_i is moved to clique partition, out of $k + 2$ non-neighbors, at most k can be deleted, and at most one could be moved to the independent partition, which leads to a contradiction to the fact that there exists a set S of size k, such that $G \setminus S$ is a split graph. Similarly, the vertices in H_c will either end up in the clique partition of the remaining split graph or will get deleted and hence will be in the solution. $\square$

The previous lemma justifies the correctness of the next reduction rule.

Reduction Rule 2. *If there is a vertex $v \in H_i \cap H_c$, then delete v and decrease k by 1.*

Lemma 6.5. *Reduction Rule 2 is correct.*

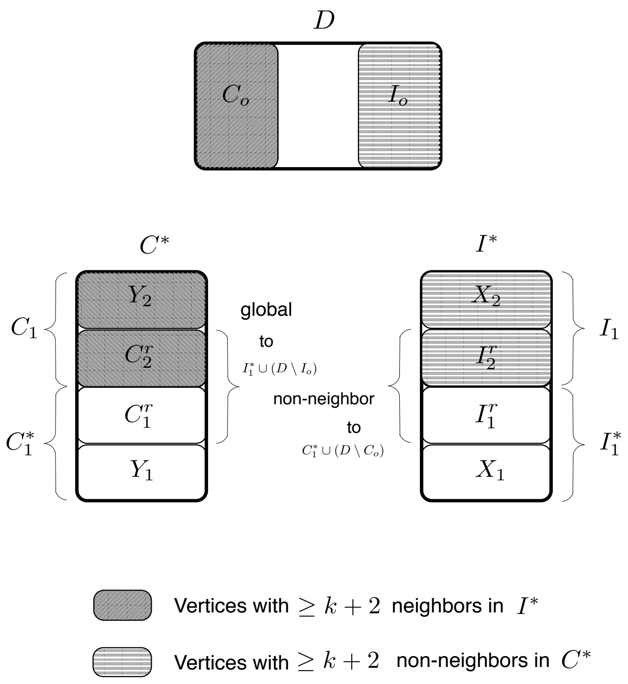
Proof. Let Z_1 be a clique non-adjacent to v and Z_2 be an independent set adjacent to v, such that $|Z_1| \geq (k+2)$ and $|Z_2| \geq (k+2)$. It is sufficient to show that v is part of every solution of size at most k. Suppose that v is not in some solution S of size at most k. Let $(C \uplus I)$ be a split partition of $G \setminus S$. We first consider the case when $v \in C$. Then, the set $Z_1 \setminus S$, which contains at least two elements, lies in I, which is not possible. Now, consider the case when $v \in I$. In this case, the set $Z_2 \setminus S$, which contains at least two elements, lies in C, which is also a contradiction. □

We now partition the vertex set of the resulting graph G as follows (see Figure 6.1 for a schematic diagram).

- Let $C_1 = H_c \cap C^*$ be the set of vertices of C^* which have high independent set neighborhood, and let $I_1 = H_i \cap I^*$, is the set of vertices of I^* which have high clique non-neighborhood.
- Similarly $C_o = H_c \cap D$, is the set of vertices of D which have high independent set neighborhood, and $I_o = H_i \cap D$, is the set of vertices of D which have high clique non-neighborhood.
- Let $C_1^* = C^* \setminus C_1$, and $I_1^* = I^* \setminus I_1$.
- Let $Y_1 \subseteq C_1^*$ and $Y_2 \subseteq C_1$ be sets of vertices which have a non-neighbor in $(D \setminus I_o) \cup I_1^*$. Let $C_1^r = C_1^* \setminus Y_1$ and $C_2^r = C_1 \setminus Y_2$ (i.e. C_1^r and C_2^r are global to $(D \setminus I_o) \cup I_1^*$).
- Let $X_1 \subseteq I_1^*$ and $X_2 \subseteq I_1$ be sets of vertices which have a neighbor in $(D \setminus C_o) \cup C_1^*$. Let $I_1^r = I_1^* \setminus X_1$ and $I_2^r = I_1 \setminus X_2$ (i.e. I_1^r and I_2^r do not have any edges to $(D \setminus C_o) \cup C_1^*$).

Before giving further reduction rules, we prove the following lemmas.

Lemma 6.6. *Let $X \subseteq C^*$, such that $|X| > k+2$ be global to $I_1^* \cup (D \setminus I_o)$. Let G' be the graph obtained by deleting all but $k+2$ vertices of X and deleting all the edges between X and $I_1 \cup I_o$. Then, (G,k) is a* YES *instance of* SVD *if and only if (G',k) is a* YES *instance of* SVD.



D
C_o
I_o
C^*
I^*
Y_2
C_2^r
C_1^r
Y_1
X_2
I_2^r
I_1^r
X_1
C_1
C_1^*
I_1
I_1^*
global
to
$I_1^* \cup (D \setminus I_o)$
non-neighbor
to
$C_1^* \cup (D \setminus C_o)$
Vertices with $\geq k+2$ neighbors in I^*
Vertices with $\geq k+2$ non-neighbors in C^*

Figure 6.1: Partitions of G.

Proof. Let $\bar{X}$ be the truncated clique. Suppose that (G, k) is a YES instance and let S be a solution to this instance. We claim that there is a solution S_1 for the instance (G, k), disjoint from X. If S itself is disjoint from X, then we are done. Suppose that S contained some vertex v of X. We simply remove this vertex from S and add it to the clique of the split partition of $G \setminus S$. Since the only vertices v is non-adjacent to, are the vertices of $I_1 \cup I_o$ and these do not lie in the clique partition of $G \setminus S$ anyway (by lemma 6.4), the resulting partition is also a split partition. Hence, we may assume that the solution S is disjoint from X. Now, we know that at most one vertex of X can lie in the independent partition of $G \setminus S$. Since this vertex does not have any non-neighbor in the clique partition of $G \setminus S$, we may move this vertex to the clique partition as well. Now, we claim that S is also a solution for the reduced instance (G', k). We have proved that there exists a split partition of $G \setminus S$ such that all the vertices of X lie in the clique. Also, since S is disjoint from X, after truncating X, the clique partition of $G \setminus S$ remains clique (we delete edges only to $I_1 \cup I_o$, which can not be in the clique side), while the independent set side is unaffected. Hence, S is also a solution for (G', k).

For the converse direction, suppose (G', k) has a solution S'. Since the vertices of $I_1 \cup I_o$

have high clique non-neighborhood in G', they are either in S', or in the final independent partition (by lemma 6.4). Analogous to the argument in the above paragraph, we can argue that $\bar{X}$ is disjoint from S' and lies in the clique partition. Now, observe that replacing $\bar{X}$ with X results in a split partition of the graph $G \setminus S'$, implying that (G, k) is a Yes instance. This concludes the proof of correctness of this reduction rule. □

The above lemma has an analogous counterpart in the case when $X \subseteq I^*$, $|X| > k+2$ and X does not have any edges to $C_1^* \cup (D \setminus C_o)$. The proof is identical, except we now consider independent sets where we considered cliques and we consider neighbors where we considered non-neighbors. We simply state the lemma without proof.

Lemma 6.7. *Let $X \subseteq I^*$ be such that $|X| > k+2$ and X does not have any edges to $C_1^* \cup (D \setminus C_o)$. Let G' be the graph obtained by deleting all but $k+2$ vertices of X and adding all the edges between X and $C_1 \cup C_o$. Then, (G, k) is a* Yes *instance of* SVD *if and only if (G', k) is a* Yes *instance of* SVD*.*

Now,we are ready to give the reduction rules, which will help us bound the size of C^* and I^*.

Reduction Rule 3. *If $|C_1^r| > k+2$, then delete all edges between C_1^r and $I_1 \cup I_o$ and delete all but $k+2$ vertices of C_1^r.*

Reduction Rule 4. *If $|C_2^r| > k+2$, then delete all edges between C_2^r and $I_1 \cup I_o$ and delete all but $k+2$ vertices of C_2^r.*

Since C_1^r and C_2^r are global to $(D \setminus I_o) \cup I_1^*)$, the correctness of these reduction rules follows immediately from lemma 6.6. Analogous to reduction rules 3 and 4, we get the following rules for the independent set side.

Reduction Rule 5. *If $|I_1^r| > k+2$, then add all edges between I_1^r and $C_1 \cup C_o$ and delete all but $k+2$ vertices of I_1^r.*

Reduction Rule 6. *If $|I_2^r| > k+2$, then add all edges between I_2^r to $C_1 \cup C_o$ and delete all but $k+2$ vertices of I_2^r.*

The correctness of these reduction rules follows from lemma 6.7. Now, towards bounding the size of the sets, we first show that the size of at least one of the sets, C_1^* and I_1^* is bounded by a linear function of k.

Lemma 6.8. *When none of the reduction rules apply, $|C_1^*| \leq 2k+2$ or $|I_1^*| \leq 2k+2$.*

Proof. As no vertex in C_1^* has high independent set neighborhood, the number of edges between the sets I_1^* and C_1^* is at most $|C_1^*|(k+1)$. Also, since no vertex in I_1^* has high clique non-neighborhood, the number of edges between the sets I_1^* and C_1^* is at least $|I_1^*|(|C_1^*| - (k+1))$. This implies that $(|C_1^*| + |I_1^*|)(k+1) \geq |C_1^*| \cdot |I_1^*|$. Substituting $|C_1^*| = 2k + c_1$ and $|I_1^*| = 2k + c_2$, where $c_1, c_2 > 2$, we get the following.

$$\begin{aligned}
&(2k + c_1 + 2k + c_2)(k+1) \geq (2k + c_1)(2k + c_2) \\
\Rightarrow\ &4k^2 + 4k + (c_1 + c_2)k + c_1 + c_2 \geq 4k^2 + 2k(c_1 + c_2) + c_1c_2 \\
\Rightarrow\ &(c_1 + c_2 - c_1c_2) + (4k - (c_1 + c_2)k) \geq 0
\end{aligned}$$

which can not be true, since for $c_1, c_2 > 2$, $(c_1 + c_2) < c_1c_2$ and $4k < (c_1 + c_2)k$ and hence we get a contradiction. $\square$

Lemma 6.9. *When none of the reduction rules apply, the number of vertices in $C_1^* \cup I_1^*$ is $\mathcal{O}(k^2)$.*

Proof. **Case 1: When $|I_1^*| \leq 2k+2$.** We observe that the size of C_1^r is already bounded by reduction rule 3, so to bound the size of $C_1^* \cup I_1^*$, we only need to bound the size of Y_1. All the vertices in Y_1 have at least one non-neighbor in $(D \setminus I_o) \cup I_1^*$. Also, we know that $(D \setminus I_o) \cup I_1^*$ has $\mathcal{O}(k)$ vertices and each such vertex has at most $\mathcal{O}(k)$ non-neighbors in C^*. Hence, the number of vertices in Y_1 is $\mathcal{O}(k^2)$. This gives a total bound of $\mathcal{O}(k^2)$ on size of $C_1^* \cup I_1^*$.

Case 2: When $|C_1^*| \leq 2k+2$. We observe that the size of I_1^r is already bounded by reduction rule 5, so to bound the size of $C_1^* \cup I_1^*$, we only need to bound the size of X_1. All the vertices in X_1 have at least one neighbor in $(D \setminus C_o) \cup C_1^*$. Also, we know that

$(D \setminus C_o) \cup C_1^*$ has $\mathcal{O}(k)$ vertices and each such vertex has at most $\mathcal{O}(k)$ neighbors in I^*. Hence, the number of vertices in X_1 is $\mathcal{O}(k^2)$. This gives a total bound of $\mathcal{O}(k^2)$ on size of $C_1^* \cup I_1^*$. □

We observe that the only unbounded sets at this point in C^* and I^* are X_2 and Y_2 respectively. All the other sets are bounded by $\mathcal{O}(k^2)$. In the next lemma, we bound the sizes of C^* and I^*.

Lemma 6.10. *When none of the reduction rules apply, the sets C^* and I^* contain $\mathcal{O}(k^3)$ vertices.*

Proof. As stated earlier, bounding the size of X_2 and Y_2 by $\mathcal{O}(k^3)$ will give us the desired result, as all the other sets in C^* and I^* are already bounded by $\mathcal{O}(k^2)$. Notice that all the vertices in Y_2 have a non-neighbor in $(D \setminus I_o) \cup I_1^*$, which has $\mathcal{O}(k^2)$ vertices. Also, any vertex in $(D \setminus I_o) \cup I_1^*$ has $\mathcal{O}(k)$ non-neighbors in C^* and $Y_2 \subseteq C^*$. This gives a bound of $\mathcal{O}(k^3)$ on size of Y_2.

Similarly, all the vertices in X_2 have a neighbor in $(D \setminus C_o) \cup C_1^*$, which has $\mathcal{O}(k^2)$ vertices. Also, any vertex in $(D \setminus C_o) \cup C_1^*$ has $\mathcal{O}(k)$ neighbors in I^* and $X_2 \subseteq I^*$. This gives a bound of $\mathcal{O}(k^3)$ on size of X_2. □

Summing up the bounds we have obtained, leads to the following theorem.

Theorem 6.5. *There is a vertex kernel for* SVD *with $\mathcal{O}(k^3)$ vertices.*

6.4 Split Edge Deletion

In this section, we present a subexponential algorithm for SED using the *Color and Conquer* approach introduced by Alon, Lokshtanov and Saurabh [ALS09]. We first design a randomized subexponential algorithm for this problem which succeeds with high probability. We then describe a way of derandomizing this algorithm to obtain a deterministic algorithm.

6.4.1 A randomized subexponential algorithm for SED

This algorithm consists of three steps. In the first step, we reduce the instance (G, k) to an equivalent instance (G', k') with $\mathcal{O}(k^2)$ vertices. In the second step, we color the vertices of the graph uniformly at random and we prove that with a sufficiently high probability, all the edges of some k-sized solution (if one exists) are non-monochromatic. Finally, we give an algorithm to check if a colored instance of SED has a non-monochromatic split edge deletion set of size at most k.

Kernelization. We first apply the kernelization algorithm (see Section 6.4.3) which, given an instance (G, k) of SED, in polynomial time, returns an equivalent instance (G', k') of SED such that the number of vertices in G' is $\mathcal{O}(k^2)$ and $k' \leq k$. In the rest of this section, we will assume that the given instance is an instance of this kind.

Probability of a Good Coloring. We now color the vertices of G independently and uniformly at random with $\sqrt{8k}$ colors and let A_c be the set of non-monochromatic edges. Suppose that $(G = (V, E), k)$ is a Yes instance and let $S \subseteq E$ be a solution to this instance. We now show that the probability of S being contained in A_c is at least $2^{-\mathcal{O}(\sqrt{k})}$. We begin by estimating the probability of obtaining a proper coloring (making all the edges non-monochromatic) when applying the above random experiment on a graph with k edges.

Lemma 6.11. ([ALS09]) *If the vertices of a graph on q edges are colored independently and uniformly at random with $\sqrt{8q}$ colors then the probability that G is properly colored is at least $(2e)^{-\sqrt{q/8}}$.*

Now, since we colored each vertex of the graph G independently, the graph induced on the set S, of size at most k, will be properly colored with probability at least $2^{-\mathcal{O}(\sqrt{k})}$, which gives us the following lemma.

Lemma 6.12. *Let $(G = (V, E), k)$ be a* Yes *instance of* SED *which is colored by the random process described above, and let $S \subset E$ be a solution for this instance. The probability that no edge in S is monochromatic is at least $2^{-\mathcal{O}(\sqrt{k})}$.*

Solving a Colored Instance. We now present an algorithm to test if there is a *colorful* (all edges non-monochromatic) split edge deletion set in a given colored instance of SED. In the colored instance, every vertex is colored with one of $\sqrt{8k}$ colors. We start with the following simple observation.

Observation 6.13. *Let $G = (V_1 \cup V_2 \cup ... \cup V_t, E)$ be a t-colored graph with color classes $V_1, \ldots, V_t$. If there exists a colorful split edge deletion set S in G, then $G[V_i]$ is a split graph for every V_i.*

We now proceed to the description of the algorithm. Suppose the given instance had a colorful split edge deletion set S. Observation 6.13 implies that $G[V_i]$ is a split graph and it remains a split graph in $G \setminus S$. Hence, we use Lemma 6.2 to enumerate the split partitions of $G[V_i]$ for each i. Fixing a split partition for each $G[V_i]$ results in a *combined* split partition for the vertices in V. There are $\mathcal{O}(k^2)$ split partitions for each V_i and $\mathcal{O}(\sqrt{k})$ such sets. Hence, there are $k^{\mathcal{O}(\sqrt{k})}$ combined split partitions. Now, it simply remains to check if there is a combined split partition $(C \uplus I)$ such that the number of edges in the graph $G[I]$ is at most k and return YES if and only if there is such a combined split partition. Hence, we have the following lemma.

Lemma 6.14. *Given a colored instance (G, k) of* SED *of size $\mathcal{O}(k^2)$, we can test if there is a colorful SED set of size at most k in time $2^{\mathcal{O}(\sqrt{k}\log k)}$.*

Combining Lemmas 6.12 and 6.14, we get the following theorem.

Theorem 6.6. *There is a randomized FPT algorithm for* SED *running in time $2^{\mathcal{O}(\sqrt{k}\log k)} + n^{\mathcal{O}(1)}$ with a success probability of at least $2^{-\mathcal{O}(\sqrt{k})}$.*

6.4.2 Derandomization with Universal Coloring Families

For integers m, k and r, a family F of functions from $[m]$ to $[r]$ is called a universal (m, k, r)-coloring family if, for any graph G on the set of vertices $[m]$ with at most k edges, there exists a function $f \in F$ which gives a proper vertex coloring of G. Suppose the kernel we obtain has size bounded by ck^2, then an explicit construction of a $(ck^2, k, \sqrt{8k})$-coloring family is known to exist.

Theorem 6.7. ([ALS09]) *There exists an explicit universal $(ck^2, k, \sqrt{8k})$-coloring family F of size at most $2^{\mathcal{O}(\sqrt{k}\log k)}$.*

Instead of the randomized coloring step of our algorithm, we can try each function in the universal coloring family given by Theorem 6.7. Hence, we have the following theorem.

Theorem 6.8. *There is an algorithm which solves* SED *in time $2^{\mathcal{O}(\sqrt{k}\log k)} + n^{\mathcal{O}(1)}$.*

6.4.3 Improved Kernel for SED

In this subsection, we use a subset of the reduction rules for SED given in [Guo07] to show the existence of a kernel with a quadratic number of vertices. The following are the reduction rules which we will apply on the given instance.

Reduction Rule 1. ([Guo07]) *Delete vertices from G which are not part of an induced subgraph isomorphic to $2K_2$, C_4 or C_5.*

From this point on, we refer to an induced subgraph isomorphic to $2K_2$, C_4 or C_5, as an induced $2K_2$, C_4 or C_5 respectively. When Reduction Rule 1 no longer applies, every vertex in G is part of some induced $2K_2$, C_4 or C_5.

Reduction Rule 2. ([Guo07]) *If two adjacent edges (u, v) and (u, w) occur together in more than k induced C_4s, then delete (u, v) and (u, w) from G and add two edges (a, v) and (b, w), where a and b are two new vertices of degree 1.*

Reduction Rule 3. ([Guo07]) *If an edge e occurs in more than k induced $2K_2$'s, then delete e from G and reduce k by one.*

We refer to [Guo07] for the correctness of these reduction rules. We apply the above reduction rules exhaustively, in the order in which they are presented, and obtain a reduced instance (G', k'). For the sake of notational convenience, we denote the reduced instance by (G, k). In the rest of this discussion, we will assume that the reduced instance is a Yes instance and prove a bound on the size of the instance with this assumption. Let S be a minimal solution with at most k edges and let $(C \uplus I)$ be a split partition of the graph

$G \setminus S$. We call a vertex of G *affected* if some edge in S is incident on it, and *unaffected* otherwise. Observe that there are at most $2k$ affected vertices in G. We now make the following important observation.

Observation 6.15. *All the affected vertices lie in the independent set I.*

Proof. Suppose there was an affected vertex in the clique partition C. Then, adding back the edges in S which are incident on this affected vertex in the clique partition also results in a split partition, which contradicts the minimality of S. □

Lemma 6.16. *Every induced C_4 in G intersects S in exactly one edge, or in exactly two adjacent edges of C_4 or in all the four edges.*

Proof. We prove the lemma by considering an induced C_4, $\{v_1, v_2, v_3, v_4\}$ and the set of affected vertices of this C_4. Since at least one edge of the C_4 has to be deleted, at least 2 vertices are affected. If exactly 2 vertices are affected, then it must be the case that exactly one edge of the C_4 is present in S. If exactly 3 vertices, say v_1, v_2 and v_3 are affected, then by Observation 6.15, these vertices lie in the independent partition and hence the edges (v_1, v_2) and (v_2, v_3) are contained in S. Since v_4 is unaffected, no other edge of this C_4 is in S. Finally, in the case when all four vertices are affected, since they all must lie in the independent partition, S contains all 4 edges of this C_4. This completes the proof of the lemma. □

Lemma 6.17. *Every induced C_5 in G intersects S in exactly two adjacent edges of C_5 or in exactly three adjacent edges of C_5 or in all the five edges.*

Proof. We fix a C_5, say $\{v_1, v_2, v_3, v_4, v_5\}$, and prove this lemma in the same way as the previous one. If only one edge, say (v_1, v_2) of the C_5 is in the solution, then the edges (v_2, v_3) and (v_5, v_1) form an induced $2K_2$, disjoint from S, which is not possible. Hence, at least two edges of the C_5 are in the solution, and at least three vertices are affected. If exactly three vertices are affected, then exactly two adjacent edges of the C_5 are in S, because we have to delete two edges from the cycle affecting only these three vertices, and this is the only possible way. Suppose exactly 4 vertices are affected. Then, by Observation 6.15, the edges between these vertices must lie in S and hence exactly 3 adjacent edges are in S. Finally, when all the vertices are affected then all the edges of the C_5 lie in S. □

We now give a bound on the number of vertices in the set I, using the following lemmas.

Lemma 6.18. *There are $\mathcal{O}(k^2)$ vertices which are part of an induced $2K_2$ in G.*

Proof. Every edge in the graph is contained in at most k induced $2K_2$'s in G (otherwise Reduction Rule 3 will be applicable). Since there is a solution of size at most k, these edges can together be contained in at most $\mathcal{O}(k^2)$ many $2K_2$'s and hence the number of vertices of G involved in an induced $2K_2$ is bounded by $\mathcal{O}(k^2)$. This completes the proof of the lemma. □

Lemma 6.19. *If $v \in I$ is an unaffected vertex, then v is not part of an induced C_4 or C_5 in G.*

Proof. Suppose that v is unaffected and v is part of a C_4. Since at least two vertices of the C_4 are affected, at least one neighbor of v on the C_4 is affected. Since this neighbor also lies in I (see Observation 6.15), the edge between these two vertices is contained in S, contradicting that v was unaffected.

Now, suppose that v is unaffected and v is part of a C_5. By Lemma 6.17), we know that at least two edges of the C_5 lie in the solution, which means at least three vertices are affected. Hence, some neighbor of v is affected, implying that v is affected as well. This completes the proof of the lemma. □

Since we have bounded the number of both affected and unaffected vertices in I, we have the following lemma.

Lemma 6.20. *There are $\mathcal{O}(k^2)$ vertices in the independent set I.*

We now proceed to bound the number of vertices inside the clique C. To do so, we introduce the notion of a *sliced* vertex. For every edge $(p, q) \in S$, and a vertex $v \in C$, we say that the edge (p, q) *slices* v if v is adjacent to p but not to q or vice versa. We say that a vertex

$v \in C$ is *sliced* if some edge in S slices it and *unsliced* otherwise. Observe that the sets of sliced and unsliced vertices form a partition of C.

Lemma 6.21. *If $v \in C$ is not sliced by any edge in S, then v is not part of an induced C_4 or C_5 in G.*

Proof. Suppose v was part of a C_4, $\{v_1, v_2, v_3, v_4\}$ where $v = v_1$. Since v is in the clique, it is unaffected (by observation 6.15) and (v_1, v_2), (v_4, v_1) are not in S. Since at least one edge from C_4 has to be in the solution, the edge (v_2, v_3) or (v_3, v_4) must be in S. But now, either of these two edges slices v, a contradiction. Suppose v was part of a C_5, $\{v_1, v_2, v_3, v_4, v_5\}$ where $v = v_1$. By Lemma 6.17, at least 2 edges of the C_5 are in S, implying that at least one of them will slice v, a contradiction. □

Since every unsliced vertex must be part of an induced $2K_2$, by Lemma 6.18 the number of unsliced vertices in the set C is bounded by $\mathcal{O}(k^2)$. To bound the number of sliced vertices in C, we argue that each edge in S can slice $\mathcal{O}(k)$ vertices of C, resulting in the following lemma.

Lemma 6.22. *There are $\mathcal{O}(k^2)$ sliced vertices in C.*

Proof. Let $e = (p, q)$ be an edge in S. By Observation 6.15, p and q are in the independent set I. Let $X(e)$ be the set of vertices in C sliced by e, $X(p) = X(e) \cap N(p)$ and $X(q) = X(e) \cap N(q)$. Then $X(e) = X(p) \uplus X(q)$. We will count the vertices of $X(e)$ as follows.

We first consider the following case.

Case 1: The sets $X(p)$ and $X(q)$ are both non-empty. We claim that in this case, $|X(p)|, |X(q)| \leq k$. Fix a vertex $w \in X(q)$. Then, for every vertex $v \in X(p)$, the vertices $\{p, q, w, v\}$ induce a C_4 in G. Hence, if there are more than k vertices in $X(p)$, then there are more than k induced C_4's in G which pairwise have the edges (p, q) and (q, w) in common. But this implies that Reduction Rule 2 applies, contradicting the irreducibility of G. Hence, we conclude that $X(p)$ must have at most k vertices. Analogously, we can bound the size of the set $X(q)$ by k.

Case 2: One of the two sets, say $X(q)$ is empty and $X(p)$ is non-empty, and suppose that there is an edge $(q, r) \in S$ such that $(p, r) \notin E$.

Since r is affected, we know that $r \in I$. Let $X_1 = X(p) \cap N(r)$ and $X_2 = X(p) \setminus X_1$. Observe that, for any vertex $v \in X_1$, the vertices $\{p, q, r, v\}$ form an induced C_4 in G. Hence, if there were more than k vertices in X_1, then there would be more than k induced C_4's in G which pairwise have only the edges (p, q) and (q, r) in common. But then Reduction Rule 2 applies, which contradicts the irreducibility of the instance. We now move on to bounding the size of the set X_2. Let u and v be two vertices in X_2 (if there are no two vertices in X_2, we already have the required bound). Since $u, v \notin N(q) \cup N(r)$ the edges (u, v) and (q, r) form an induced $2K_2$ in G. Hence, if the number of vertices in X_2 exceeds $2\sqrt{k} + 1$, then there would be more than k induced $2K_2$'s in G which have the edge (q, r) in common. But in this case, Reduction Rule 3 applies, a contradiction. Hence, the set $X(e)$ contains at most $2k$ vertices.

Now, we first try to associate every sliced vertex of C to some edge satisfying the premise of Case 1 or 2. We have already argued that there are $\mathcal{O}(k^2)$ such vertices.

Now we bound the remaining sliced vertices by showing that they cannot be part of any induced C_4 or C_5 of G, and hence can only be part of $2K_2$s, and hence they add up to $\mathcal{O}(k^2)$ in number.

Consider a vertex v sliced by an edge $(p, q) \in S$, $v \in X(p)$ and has not been associated with an edge satisfying Case 1 or Case 2. Suppose that v was part of a C_4 $\{v_1, v_2, v_3, v_4\}$ where $v = v_1$. Suppose that the edge (v_2, v_3) is in S. Now, if the vertex v_4 is in C, then v_1 and v_4 are sliced by the edge (v_2, v_3), and hence we will be in the case (Case 1) when $X(p)$ and $X(q)$ are non-empty where $(p, q) = (v_2, v_3)$. Similarly, if v_4 is in I, then we will be in the case (Case 2) when even if $X(p)$ or $X(q)$ is empty, there is an edge $(q, r) = (v_3, v_4)$ in G, such that there is no edge $(p, r) = (v_2, v_4)$.

Similarly, we can show that v cannot be part of an induced C_5. Hence, it must be the case that v is part of an induced $2K_2$. Recall that we have already bounded the number of such vertices by $\mathcal{O}(k^2)$. $\square$

We have thus bounded the number of vertices in C and I, and hence bounded the number of vertices of the graph G, leading to the following theorem.

Theorem 6.9. *There is a kernel for* SED *with* $\mathcal{O}(k^2)$ *vertices.*

6.5 Conclusion

In this chapter we studied the parameterized complexity of deleting k edges/vertices to get to the class of split graphs. We obtained faster parameterized algorithms as well as smaller sized kernels for these problems. Cygan and Pilipczuk [CP13] have subsequently improved one of the four results presented in this chapter. That is, SVD can be solved in time $\mathcal{O}^*(1.2738^k k^{\mathcal{O}(\log k)})$. Finally, an interesting project could be to systematically identify other parameterized problems that admit subexponential parameterized algorithms on general graphs.

Chapter 7

Parameterized Algorithms for Almost Forest Deletion

7.1 Introduction

The Feedback Vertex Set (FVS) problem has been widely studied in the field of parameterized algorithms. A series of results have improved the running times to $\mathcal{O}^*(3.619^k)$ in deterministic setting [KP14] and $\mathcal{O}^*(3^k)$ in randomized setting [CNP+11]. In this chapter, we look at a generalization of the FVS where we are allowed to delete both vertices and edges. Here, we want to delete at most k vertices *and* ℓ edges from the input graph to get to a forest. The problem can also be looked at as an instance of the vertex deletion problem where we want to delete at most k vertices such that the resulting graph is at most ℓ edges away from being a forest. We call such graphs ℓ-forests and the problem is called Almost Forest Deletion. The aim of this chapter is to generalize the techniques used for FVS to get FPT algorithms and polynomial kernels for Almost Forest Deletion, when parameterized by both k and ℓ.

Our results. We show that Almost Forest Deletion can be solved in time $\mathcal{O}^*(5.0024^{(k+\ell)})$. We arrive at the result using the iterative compression technique which was introduced in [RSV04] and a non-trivial measure which helps us in getting the desired running time. Then we explore the kernelization complexity of the problem, and show that Almost

Forest Deletion admits a polynomial kernel with $\mathcal{O}(k\ell(k+\ell))$ edges. For arriving at the result, we first make use of Expansion Lemma and Gallai's theorem for reducing the maximum degree of the graph, and then we bound the size of the graph. It is easy to see that for a Yes instances (G,k,ℓ) of Almost Forest Deletion, the treewidth of G is bounded by $k+\ell$. Since we have an algorithm of the form $\mathcal{O}^*(c^{(k+\ell)})$ on general graphs, the question of finding an $\mathcal{O}^*(c^{\mathbf{tw}})$ algorithm becomes interesting for bounded treewidth graphs. We answer this question affirmatively by giving an $\mathcal{O}^*(17^{\mathbf{tw}})$ algorithm for graphs which come with a tree decomposition of width tw. This algorithm, along with the notion of bidimensionality gives rise to an algorithm for Almost Forest Deletion on planar graphs running in time $2^{\mathcal{O}(\sqrt{\ell+k})}n^{\mathcal{O}(1)}$. Our methods are based on the known methods to solve the Feedback Vertex Set problem.

7.2 Preliminaries

A *k-flower* in a graph is a set of k cycles which are vertex disjoint except for one vertex v, which is shared by all the cycles in the set. The vertex v is called *center* of the flower and the cycles are called the *petals* of the flower. An ℓ-forest is a graph which is at most ℓ edges away from being a forest, i.e. the graph can be transformed into a forest by deleting at most ℓ edges. For a connected component C of a graph, we call the quantity $|E(G[C])|-|C|+1$ the *excess of* C and denote it by $\mathsf{ex}(C)$. It can also be equivalently defined as the minimum number of edges we need to delete from the connected component to get to a tree. For a graph G, let $\mathcal{C}$ be the set of its connected components. We define the excess of the graph, $\mathsf{ex}(G)$ as follows.

$$\mathsf{ex}(G)=\sum_{C\in\mathcal{C}}\mathsf{ex}(C)$$

As in the case of components, this measure can be equivalently defined as the minimum number of edges we need to delete from G to to get to a forest. It is easy to see that a graph G is an ℓ-forest if and only if $\mathsf{ex}(G)\le\ell$. For $X\subseteq V(G)$ such that $G-X$ is an ℓ-forest, we call X an ℓ-forest deletion set of G. We define the Almost Forest Deletion problem as follows.

Almost Forest Deletion	**Parameter(s):** ℓ, k
Input: A graph G, integers k and ℓ.	
Question: Does there exist $X \subseteq V(G)$ such that $|X| \le k$ and $G - X$ is an ℓ-forest?	

Observation 7.1. *Let G' be a subgraph of G. If G is an ℓ-forest, then so is G'.*

Observation 7.2. *If G is an ℓ-forest, it has at most $V(G) - 1 + \ell$ edges.*

Lemma 7.3. *Let G be a graph. If there exists a vertex v such that v is not part of any cycle in G, then $\mathsf{ex}(G - \{v\}) = \mathsf{ex}(G)$. Furthermore, if v is part of a cycle in G, then $\mathsf{ex}(G - \{v\}) \le \mathsf{ex}(G) - 1$.*

Proof. Let $G' = G - \{v\}$. Let $\mathcal{C}$ be the set of connected components of G and let C be the connected component of G containing v. Let us first look at the case when v is not part of any cycle in G. Then all the edges incident to v are to different connected components of G'. Hence, G' has exactly d components, in which some vertex was adjacent to v in G, where d is degree of v. Let this set of connected components be $\{C_1, C_2, \ldots, C_d\}$. We observe the following.

$$\begin{aligned}
\mathsf{ex}(C) &= |E(G[C])| - |C| + 1 \\
&= \left(\sum_{i\in[d]} |E(G[C_i])|\right) + d - \left(\sum_{i\in[d]} |C_i|\right) - 1 + 1 \\
&= \sum_{i\in[d]} (|E(G[C_i])| - |C_i| + 1) \\
&= \sum_{i\in[d]} \mathsf{ex}(C_i)
\end{aligned}$$

For all the other components, the excess remains the same, and hence the excess of G remains same as that of G'. On the other hand, if v is part of a cycle, then it has at least two edges to a component of G'. Hence, the number of connected components in G' in which some vertex was adjacent to v is strictly less than the degree of v. Let these set of connected components be $\{C_1, C_2, \ldots, C_{d'}\}$, where $d' < d$. We observe the following.

$$\begin{aligned}
\mathsf{ex}(C) &= |E(G[C])| - |C| + 1 \\
&= \left(\sum_{i\in[d']} |E(G[C_i])|\right) + d - \left(\sum_{i\in[d']} |C_i|\right) - 1 + 1 \\
&\geq \left(\sum_{i\in[d']} (|E(G[C_i])| - |C_i| + 1)\right) + 1 \qquad \text{(since } d' \leq d-1\text{)} \\
&\geq \left(\sum_{i\in[d']} \mathsf{ex}(C_i)\right) + 1
\end{aligned}$$

Since all other connected components remain the same, the excess of G decreases by at least 1 by deleting a vertex which is part of a cycle. This concludes the proof of the lemma. □

Lemma 7.4. *Let $X \subseteq V(G)$ be a set of vertices of G which do not belong to any cycle. Then, G is an ℓ-forest if and only if $G - X$ is an ℓ-forest.*

Proof. We can apply Lemma 7.3 iteratively to prove this lemma. We can delete vertices of X one by one, and the excess of the graph remains the same. We continue till all the vertices of X are deleted. By application of Lemma 7.3 at every step, we have that $\mathsf{ex}(G) = \mathsf{ex}(G - X')$, where X' is set of vertices deleted till that step. Hence, G is an ℓ-forest if and only if $G - X$ is. □

Lemma 7.5. *Any ℓ-forest can have at most ℓ edge disjoint cycles.*

Proof. Let the graph G have ℓ' edge disjoint cycle such that $\ell' > \ell$. To get a forest as a subgraph of G, we must delete at least one edge from every cycle. Since the cycles are edge disjoint, we need to delete at least $\ell' > \ell$ edges from G to get to a forest. This shows that G is not an ℓ-forest. □

For this chapter, we also need to generalize the notion of kernelization for when we have multiple parameters.

Kernelization. A *kernelization* algorithm for a parameterized language L is a polynomial time procedure which takes as input an instance $(x, k_1, \ldots, k_l)$, where k_i's are the parameters and returns an instance $(x', k'_1, \ldots, k'_l)$ such that $(x, k_1, \ldots, k_l) \in L$ if and only if $(x', k'_1, \ldots, k'_l) \in L$ and $|x'| \leq h(k_1, \ldots, k_l)$ and $k'_i \leq g(k_1, \ldots, k_l)$ for all $i \in [l]$, for some computable functions h, g. The returned instance is said to be a *kernel* for L and the function h is said to be the *size of the kernel.*

7.3 An $O^*(c^{(k+\ell)})$ algorithm for Almost Forest Deletion

In this section we will present a $c^{(\ell+k)}n^{\mathcal{O}(1)}$ algorithm for Almost Forest Deletion. We use the well known technique of iterative compression and arrive at the desired running time after defining a non-trivial measure.

Given an instance (G, k, ℓ) of Almost Forest Deletion, let $V(G) = \{v_1, \ldots, v_n\}$ and define vertex sets $V_i = \{v_1, \ldots, v_i\}$, and let the graph $G_i = G[V_i]$. We iterate through the instances (G_i, k, ℓ) starting from $i = k + 1$. For the i^{th} instance, we try to find an ℓ-forest deletion set $\hat{S}_i$ of size at most k, with the help of a *known* ℓ-forest deletion set S_i of size at most $k + 1$. Formally, the compression problem we address is the following.

Almost Forest Deletion Compression **Parameter(s):** k, ℓ

Input: A graph G, and ℓ-forest deletion set of G of size at most $k+1$, integers k and ℓ.

Question: Does there exist an ℓ-forest deletion set S for G of size at most k?

Lemma 7.6. *If* Almost Forest Deletion Compression *can be solved in $f(k, \ell)n^c$ time, then* Almost Forest Deletion *can be solved in $f(k, \ell)n^{c+1}$ time.*

Proof. We solve the Almost Forest Deletion problem by iteratively solving at most $n - k$ instances of the Almost Forest Deletion Compression problem as follows. Let $I_i = (G_i, S_i, k, \ell)$ be the i^{th} instance. Clearly, the set V_{k+1} is an ℓ-forest deletion set of size at most $k + 1$ for the instance I_{k+1}. It is also easy to see that if $\hat{S}_{i-1}$ is an ℓ-forest deletion set of size at most k for instance I_{i-1}, then the set $\hat{S}_{i-1} \cup \{v_i\}$ is an ℓ-forest deletion set of size at most $k + 1$ for the instance I_i. We use these two observations to start of the iteration with the instance $(G_{k+1}, S_{k+1} = V_{k+1}, k, \ell)$ and look for an ℓ-forest deletion set of size at most k for this instance. If there is such an ℓ-forest deletion set $\hat{S}_{k+1}$, we set

$S_{k+2} = \hat{S}_{k+1} \cup \{v_{k+2}\}$ and ask for an ℓ-forest deletion set of size at most k for the instance I_{k+2} and so on. It is easy to verify that if, during any iteration, the corresponding instance does not have an ℓ-forest deletion set of the required size, it implies that the original instance is also a NO instance. This follows from the fact that if a graph G has an ℓ-forest deletion set of size at most k, then any vertex induced subgraph of G also has a ℓ-forest deletion set of size at most k. Finally, the solution for the original input instance will be $\hat{S}_n$. Since there can be at most n iterations, the total time taken to solve the original instance is bounded by n times the time required to solve the ALMOST FOREST DELETION COMPRESSION problem. □

For designing an algorithm for ALMOST FOREST DELETION COMPRESSION, let the input instance be (G, S, k, ℓ). We guess a subset $Y \subseteq S$ with the intention of picking these vertices in our hypothetical solution for this instance and not picking the rest of the vertices in S in the solution. We delete the set Y from the graph and decrease k by $|Y|$. We then check if the graph $G[S \setminus Y]$ is an ℓ-forest and if it is not, then reject this guess of Y as a spurious guess. Suppose that $G[S \setminus Y]$ is indeed an ℓ-forest. Hence, it now remains to check if there is an ℓ-forest deletion set S' of the size $k' = k - |Y|$ which is disjoint from $S \setminus Y$, and $G - (Y \cup S')$ is an ℓ-forest. More precisely, we have an instance of ALMOST FOREST DELETION DISJOINT COMPRESSION (AFDDC), which is defined as follows.

AFDDC **Parameter(s):** k, ℓ

Input: A graph G, an ℓ-forest deletion set S of G, integers k and ℓ

Question: Does there exist an ℓ-forest deletion set of G disjoint from S of size at most k?

To solve the problem, we first design a set of reduction rules, and prove some lemmata about their correctness.

Reduction Rule 1. *If there exists a vertex v of degree at most one in the graph, delete it. x*

Reduction Rule 2. *If there exists $v \in V(G) \setminus S$ such that $G[S \cup \{v\}]$ is not an ℓ-forest, delete v and decrease k by* 1.

Reduction Rule 3. *If there exists a vertex $v \in V(G) \setminus S$ of degree two, such that at least one of its neighbours are in $V(G) \setminus S$, then delete v and put a new edge between its neighbours (even if they were already adjacent). If both of v's edges are to the same vertex, delete v and put a new self loop on the adjacent vertex (even if it has self loop(s) already).*

Now we proceed towards proving the correctness of these reduction rules. Let (G, S, k, ℓ) be the instance on which the reduction rule is being applied and let (G', S', k', ℓ') be the instance after the application of the reduction rule. We say that a reduction rule is *correct* if (G, S, k, ℓ) is a Yes instance of AFDDC if and only if (G', S', k', ℓ') is a Yes instance. The correctness of Reduction Rule 1 is obvious from Lemma 7.3.

Lemma 7.7. *Reduction Rule 2 is correct.*

Proof. For showing correctness of this reduction rule, we just need to show that v is part of every ℓ-forest deletion set of G of size at most k, which is disjoint from S. This is indeed true, because if not so, then we know that $G[S \cup \{v\}]$ is not an ℓ-forest. Hence for any set X of size at most k disjoint from S which does not contain v, $G - X$, being a supergraph of $G[S \cup \{v\}]$, is not an ℓ-forest as well. □

Lemma 7.8. *Reduction Rule 3 is correct.*

Proof. Let (G, S, k, ℓ) be a Yes instance of AFDDC, and let X be an ℓ-forest deletion set of G of size at most k, which is disjoint from S. Let v be the vertex of degree 2 being deleted from the graph. Let us first examine the case where v has two distinct neighbours x and y. Without loss of generality, let $x \in V(G) \setminus S$. If $X \cap \{v, x, y\} \neq \emptyset$, then we put $X' = (X \setminus \{v\}) \cup \{x\}$ if $v \in X$ or $X' = X$ if $v \notin X$. It is easy to see that $|X'| \leq k$ and $G' - X'$ is a subgraph of $G - X$ in both the cases and hence $G' - X'$ is an ℓ-forest. On the other hand, if none of v, x or y are in X, then we look at the connected component of $G' - X$ containing x and y. This is the only connected component for which the excess could have possibly changed. But this connected component loses one vertex and two edges, while gains one new edge. So, the excess for this component also remains the same, and hence $G' - X$ is an ℓ-forest. The case when both of v's edges are to the same vertex is similar.

For the converse, let (G', S, k, ℓ) be a YES instance of AFDDC which we get after applying Reduction Rule 3 and let X be an ℓ-forest deletion set of G' of size at most k, which is disjoint from S. If X contains any neighbour of v, then in $G - X$, v has degree at most 1 and hence the edge incident on it (if it survives) does not contribute to excess of any connected component. So, $G - X$ remains to be an ℓ-forest. In the case where none of v's neighbours belong to X, only the connected component including the neighbours can possibly gain from introducing v back into the graph. But in this case, since we delete the edge between the neighbours (or the self loop) to get to G, the component gains exactly one vertex and one edge, and hence even in this case $G - X$ is an ℓ-forest. So, (G, S, k, ℓ) is a YES instance of AFDDC. This concludes the proof of the lemma. □

It is easy to see that after the exhaustive applications of the reduction rules, if there exists a vertex of degree at most one in $G - S$, then it has at least two neighbours in S.

Now we are ready to describe our algorithm for AFDDC. Given an input instance (G, S, k, ℓ) of AFDDC, we first apply the reduction rules 1-3 exhaustively. If $k < 0$, then we return that the given instance is a NO instance.

Now, we look for a vertex v of degree at most one in $G - S$ and we branch by either including v in our solution or excluding it. More precisely, we call the algorithm recursively on $(G - \{v\}, S, k - 1, \ell)$ and $(G, S \cup \{v\}, k, \ell)$. If one of the recursive call returns YES then we say that the instance was a YES instance. If there does not exist a vertex of degree at most 1 in $G - S$, then there must be a vertex v which is part of a cycle. In this case we branch on this vertex, and call the algorithm recursively on $(G - \{v\}, S, k - 1, \ell)$ and $(G, S \cup \{v\}, k, \ell)$ as we did in the previous case. This concludes the description of the algorithm. The correctness of the algorithm follows from the correctness of reduction rules and the fact that the branching is exhaustive.

To analyze the running time of the algorithm, we define a measure $\phi(I)$ for the input instance $I = (G, S, k, \ell)$ as follows.

$$\phi(I) = \alpha k + \beta \mathsf{cc}(S) + \gamma(\ell - \mathsf{ex}(G[S])) + \delta(\mathsf{ex}(G - S))$$

Here, $\mathsf{cc}(S)$ denotes the number of connected components of $G[S]$, α, β, γ and δ are positive constants such that $\delta > \beta$. We will assume these properties for now, and will fix the values of these constants later.

Lemma 7.9. *None of the reduction rules 1-3 increase the measure $\phi(I)$.*

Proof. Reduction Rule 1 deletes a vertex of degree at most one from the graph, and by Lemma 7.3, it can not affect $\mathsf{ex}(G[S])$ or $\mathsf{ex}(G - S)$. It might decrease the number of connected components in $G[S]$, but the measure $\phi(I)$ does not increase. Reduction Rule 2 deletes a vertex v for which $G[S \cup \{v\}]$ is not an ℓ-forest, and decreases k by one. So, it does not change $\mathsf{cc}(S)$ or $\mathsf{ex}(G[S])$ while it might decrease $\mathsf{ex}(G - S)$. Hence, the measure drops by at least α. For Reduction Rule 3, k, $\mathsf{cc}(S)$ and $\mathsf{ex}(G[S])$ remain the same. To see that $\mathsf{ex}(G - S)$ also remains the same, we observe that the connected component which contained v can only possibly increase its excess. But it loses exactly one vertex and one edge, hence the excess of the connected component and $\mathsf{ex}(G - S)$ remain same after application of the reduction rule. So, we have that none of the reduction rules increases the measure. $\square$

Lemma 7.10. AFDDC *can be solved in time $\mathcal{O}^*((4.0024)^k(5.0018)^\ell)$.*

Proof. We first look at the branching steps to analyse the running time of the algorithm.

1. Branching on a vertex v of degree at most 1 in $G - S$. In one branch, we call the algorithm on $(G - \{v\}, S, k - 1, \ell)$. It is easy to see that in this branch $\mathsf{cc}(S)$, $\mathsf{ex}(G[S])$ and $\mathsf{ex}(G - S)$ remain the same while k decreases by 1, and hence the measure drops by α. In the other branch, we call the algorithm recursively on $(G, S \cup \{v\}, k, \ell)$. Let $S' = S \cup \{v\}$. Since $G - S'$ is a subgraph of $G - S$, $\mathsf{ex}(G - S') \leq \mathsf{ex}(G - S)$. We know that v has at least 2 neighbours in S. If at least 2 of them belong to different components of $G[S]$, then $\mathsf{cc}(S') \leq \mathsf{cc}(S) - 1$ (in this case $\mathsf{ex}(G[S])$ might also increase, which is good) and hence the measure drops by at least β. Otherwise, they all belong to the same connected component, and hence $\mathsf{cc}(S') = \mathsf{cc}(S)$, but then by Lemma 7.3, $\mathsf{ex}(G[S']) \geq \mathsf{ex}(G[S]) + 1$. Hence the measure drops by at least γ. Also, $\mathsf{ex}(G[S'])$ does not exceed ℓ, because then we would have applied Reduction Rule 14. Hence, in this case, in one branch the measure drops by

α, while in other it drops by β or γ, but remains non-negative. This gives us two possible branching factors (α, β) and (α, γ).

2. Branching on a vertex v which is part of a cycle in $G - S$. In one branch, we call the algorithm on $(G - \{v\}, S, k - 1, \ell)$. So k decreases by 1 while $\mathsf{ex}(G[S])$ and $\mathsf{cc}(S)$ remain the same. But since v is part of a cycle in $G - S$, $\mathsf{ex}(G - (S \cup \{v\})) \leq \mathsf{ex}(G - S) - 1$ by Lemma 7.3. Hence, in this branch, the measure decreases by $\alpha + \delta$. In the other branch, we call the algorithm on $(G, S \cup \{v\}, k, \ell)$. So k remains the same while $\mathsf{ex}(G[S])$ may increase (but won't become more than ℓ), which is good. Also, $\mathsf{ex}(G - (S \cup \{v\})) \leq \mathsf{ex}(G - S) - 1$ by Lemma 7.3. Now, the problematic thing is that $\mathsf{cc}(G[S])$ may increase by 1. Hence, we get a net decrease of $\delta - \beta$. But since we have assumed $\delta > \beta$, this decrease is positive. Hence, the measure drops in every branch and remains non-negative. In this case, we get a branching factor of $(\alpha + \delta, \delta - \beta)$.

We know that $|S| \leq k$ and hence the number of connected components of $G[S]$ is bounded by k. We also know that $0 \leq \mathsf{ex}(G[S]) \leq \ell$ initially, otherwise we would have discarded the set S as an spurious set. Also, since S is an ℓ-forest deletion set of G, $\mathsf{ex}(G - S) \leq \ell$. This proves that $0 \leq \phi(I) \leq (\alpha + \beta)k + (\gamma + \delta)\ell$ initially. Also, we have seen that none of the reduction rules increase the measure. It is easy to see that the reduction rules can be applied in polynomial time.

For the branching part, we have three branching factors, (α, β), (α, γ) and $(\alpha + \delta, \delta - \beta)$. We ran a numerical program to to find values of α, β, γ and δ, which optimize the running time of the algorithm. Putting $\alpha = 1.45$, $\beta = 1.35$, $\gamma = 1.35$ and $\delta = 1.9$ gives us the branching factors of $(1.45, 1.35)$, $(1.45, 1.35)$ and $(3.35, 0.55)$, out of which $(1.45, 1.35)$ is the worst and gives us running time of $(4.0024)^k (5.0018)^\ell n^{\mathcal{O}(1)}$. $\square$

Given Lemma 7.10, the algorithm for Almost Forest Deletion Compression runs in time $\mathcal{O}(\sum_{i=0}^{k} \binom{k+1}{i} \cdot (4.0024)^i (5.0018)^l \cdot n^{\mathcal{O}(1)}) = \mathcal{O}^*(5.0024^{(k+\ell)})$. Here, the factor of $\binom{k+1}{i}$ is for the guesses we make for the set S. Finally applying Lemma 7.6, we get the following theorem.

Theorem 7.1. Almost Forest Deletion *can be solved in* $\mathcal{O}^*(5.0024^{(k+\ell)})$ *time.*

7.4 $\mathcal{O}(k\ell(k+\ell))$ kernel for Almost Forest Deletion

In this section, we give the kernelization algorithm for Almost Forest Deletion. First we give a set of reduction rules and then prove the lemmata which help us bound the size of the output instance. Throughout the section, we apply the reduction rules in order, that is, while applying a reduction rule we assume that all the reduction rules stated previously in the section have been applied exhaustively.

Reduction Rule 4. *If there exists a vertex v of degree at most one in the graph, delete it.*

Reduction Rule 5. *If there exists a vertex $v \in V(G)$ of degree two then delete v and put a new edge between its neighbours (even if they were already adjacent). If both of v's edges are to the same vertex, delete v and put a new self loop on the adjacent vertex (even if it has self loop(s) already).*

Reduction Rule 6. *If any edge has multiplicity more that $\ell + 2$, then delete all but $\ell + 2$ copies of that edge.*

Notice that reduction rules 4-6 leave the parameters k and ℓ unchanged. Let (G, k, ℓ) be the instance before one of the reduction rules 4-6 is applied and let (G', k, ℓ) be the output instance after application of the reduction rule. Correctness of Reduction Rule 4 follows from Lemma 7.4 and proof of correctness of Reduction Rule 5 is very similar to proof of correctness of Reduction Rule 3. We just do not need to care about the set S in the proof, and everything else remains the same. We prove the correctness of the Reduction Rule 6.

Lemma 7.11. *Reduction Rule 6 is correct.*

Proof. Let (u, v) be the edge with multiplicity more than $\ell + 2$ in G. Since G' is a subgraph for G, any solution X for G of size at most k will also be a solution for G'. So, if (G, k, ℓ) is a Yes instance, then so is (G', k, ℓ). For the converse, let (G', k, ℓ) be a Yes instance. Any ℓ-forest deletion set X for G' must contain either u or v, otherwise we have that $G[\{u, v\}]$ is not an ℓ-forest, which is a subgraph of $G - X$, and hence $G - X$ is also not an ℓ-forest. Since we have only altered edges between u and v and X contains at least one of them, $G - X$ is identical to $G' - X$. Hence, $G - X$ is an ℓ-forest and (G, k, ℓ) is a Yes instance. □

Given an instance (G, k, ℓ) of ALMOST FOREST DELETION, we apply reduction rules 4-6 exhaustively. Observe that after the application of these reduction rules, the graph has degree at least 3, as all the vertices of degrees 1 and 2 are taken care of by Reduction Rule 4 and Reduction Rule 5 respectively.

We prove the following lemma which talks about ℓ-forest deletion sets of graphs of bounded degree with minimum degree 3.

Lemma 7.12. *If a graph G has minimum degree at least 3, maximum degree at most d, and an ℓ-forest deletion set of size at most k, then it has less than $2\ell + k(d+1)$ vertices and less than $2kd + 3\ell$ edges.*

Proof. Let X be an ℓ-forest deletion set of G of size at most k. Let $Y = V(G) \setminus X$. We first observe the following by summing up the degrees of vertices of Y.

$$3|Y| \leq 2(|E(G[Y])| + |E(X, Y)|$$

But we know that $|E(G[Y])| \leq |Y| + \ell - 1$ and $|E(X, Y)| \leq kd$. Putting these in the above inequality, we get the following.

$$\begin{aligned} 3|Y| &\leq 2(|Y| + \ell - 1) + kd \\ |Y| &< 2\ell + kd \end{aligned}$$

For bounding $|V(G)|$, we have the following.

$$\begin{aligned} |V(G)| &\leq |X| + |Y| \\ |V(G)| &< 2\ell + k(d+1) \end{aligned}$$

For bounding the number of edges, since $|Y| < 2\ell + kd$, we know that $|E(G[Y])| <$

$2\ell + kd + \ell = kd + 3\ell$. Now we can bound $|E[G]|$ as following.

$$\begin{aligned} |E[G]| &\leq |E(G[X])| + |E(X,Y)| + |E(G[Y])| \\ &< kd + kd + 3\ell \\ &= 2kd + 3\ell \end{aligned}$$

This conclude s the proof of the lemma. $\square$

Lemma 7.12 gives rise to the following reduction rule immediately.

Reduction Rule 7. *After the application of reduction rules 4, 5 and 6 exhaustively, if either $|V(G)| \geq 2\ell + k(d+1)$ or $|E(G)| \geq 2kd + 3\ell$, where d is the maximum degree of the graph, return that the given instance is a* No *instance.*

After this, all that is left is to reduce the maximum degree of the graph. If after the exhaustive application of reduction rules 4, 5 and 6, the maximum degree of the graph is already bounded by $(k + \ell)(3\ell + 8)$ then we already have a kernel with $\mathcal{O}(k\ell(k + \ell))$ vertices and $\mathcal{O}(k\ell(k + \ell))$ edges. Hence we assume, for the rest of the section, that there exists a vertex with degree greater than $(k + \ell)(3\ell + 8)$. We need one more reduction rule before we proceed further.

Reduction Rule 8. *If there is a vertex v with more than ℓ self loops, delete v and decrease k by* 1.

Correctness of the reduction rule follows immediately from Lemma 7.5. We now try to reduce the high degree vertices. The idea is that either a high degree vertex participates in many cycles (and contributes many excess edges) and hence should be part of the solution, or only a small part of its neighbourhood is relevant for the solution. We formalize these notions by use of Gallai's theorem to find flowers and applying a set of reduction rules. Given a set $T \subseteq V(G)$, by T-path we mean a path of positive length with both endpoints in T.

Theorem 7.2 (Gallai, [Gal64])**.** *Given a simple graph G, a set $T \subseteq V(G)$ and an integer s, one can in polynomial time find either*

- *a family of $s + 1$ pairwise vertex-disjoint T-paths, or*
- *a set B of at most $2s$ vertices, such that in $G - B$ no connected component contains more than one vertex of T.*

We would want to have the neighborhood of a high degree vertex as the set T for applying Gallai's theorem and for detecting flowers. But we need to be careful, as the graph in its current form contains multiple edges and self loops. Let v be a vertex with high degree. The vertices in $N(v)$ which have at least two parallel edges to v can be greedily picked to form a petal of the flower. Let L be the set of vertices in $N(v)$ which have at least two parallel edges to v.

Reduction Rule 9. *If $|L| > k + \ell$, delete v and decrease k by 1.*

Lemma 7.13. *Reduction Rule 9 is correct.*

Proof. The correctness of the reduction rule follows from the fact that any ℓ-forest deletion set of size at most k must delete v. This is true because otherwise, we have at least $\ell + k + 1$ petals which form edge disjoint cycles, and hence after deleting any of the k vertices from $N(v)$, we will be left with at least $\ell + 1$ edge disjoint cycles. So, by Lemma 7.5, the graph will not be an ℓ-forest. □

Let $\widehat{G}$ be the graph $G - L$ with all parallel edges replaced with single edges, and all self loops removed. Now we apply Gallai's theorem on $\widehat{G}$ with $T = N(v)$ and $s = k + \ell - |L|$. If the theorem returns a collection of vertex disjoint T-paths, then it is easy to see that they are in one to one correspondence with cycles including v, and hence can be considered petals of the flower centered at v.

Reduction Rule 10. *If the application of Gallai's theorem returns a flower with more than s petals, then delete v and decrease k by 1.*

Lemma 7.14. *Reduction Rule 10 is correct.*

Proof. Let us assume that the application of Gallai's theorem on $\widehat{G}$ returns a flower centered at v with at least $k + \ell + 1 - |L|$ petals. Since $\widehat{G}$ does not contain L, we can add a pair of parallel edges of the form (x, v) for all $x \in L$ to form a flower centered at v with at least $k + \ell + 1$ petals in G. This forces v to be part of any ℓ-forest deletion set of G of size at most k. $\square$

We now deal with the case when the application of Gallai's theorem returns a set B of at most $2(k + \ell - |L|)$ vertices, such that in $\widehat{G} - B$ no connected component contains more than one vertex of T. Let $Z = B \cup L$. Clearly, $|Z| \leq 2(k + \ell) - |L|$. Now we look at the set of connected components of $\widehat{G} - (Z \cup \{v\})$. Let us call this set C. We first prove that if too many connected components in C have a cycle, then we can say that the instance is a No instance.

Reduction Rule 11. *If more than $k + \ell$ components of C contain a cycle, then return that the instance is a* No *instance.*

Lemma 7.15. *Reduction rule 11 is correct.*

Proof. All the cycles in different connected components are vertex disjoint. Deleting any set X of size at most k leave at least $\ell + 1$ cycles in the graph which are vertex disjoint and hence edge disjoint. By Lemma 7.5, we know that $G - X$ is not an ℓ-forest for any set X of size at most k, and hence (G, k, ℓ) is a No instance. $\square$

Lemma 7.16. *After applying reduction rules $4 - 11$ exhaustively, there are at least $2(\ell + 2)(k + \ell)$ components in C which are trees and connected to v with exactly one edge.*

Proof. The number of self loops on v is bounded by ℓ due to Reduction Rule 8. Number of edges from v to Z is bounded by $|B| + (\ell + 2)|L| \leq 2(k + \ell - |L|) + (\ell + 2)|L| = 2(k + \ell) + l|L| \leq (k + \ell)(\ell + 2)$. As degree of v is greater than $(k + \ell)(3\ell + 8)$, at least $(k+\ell)(3\ell+8) - (k+\ell)(\ell+2) - \ell \geq (k+\ell)(2\ell+5)$ connected components in C have exactly one vertex which is is neighbour of v. Out of these, the number of connected components containing cycles is bounded by $k + \ell$ by Reduction Rule 11. Hence, at least $2(\ell+2)(k+\ell)$ connected components are trees and are connected to v by exactly one edge. $\square$

Before we proceed further, we state the Expansion Lemma. Let G be a bipartite graph with vertex bipartition (A, B). For a positive integer q, a set of edges $M \subseteq E(G)$ is called a q-expansion of A into B if every vertex of A is incident with exactly q edges of M, and exactly $q|A|$ vertices in B are incident to M.

Lemma 7.17 (Expansion Lemma, [FLM+11])**.** *Let $q \geq 1$ be a positive integer and G be a bipartite graph with vertex bipartition (A, B) such that $|B| \geq q|A|$ and there are no isolated vertices in B. Then there exist nonempty vertex sets $X \subseteq A$ and $Y \subseteq B$ such that there is a q-expansion of X into Y and no vertex in Y has a neighbor outside X, that is, $N(Y) \subseteq X$. Furthermore, the sets X and Y can be found in time polynomial in the size of G.*

Let D the set of connected components of C which are trees and connected to v with exactly one edge. We have shown that $|D| \geq 2(\ell+2)(k+\ell)$. Now we construct an auxiliary bipartite graph H as follows. In one partition of H, we have a vertex for every connected component in D, and the other partition is Z. We put an edge between $A \in D$ and $v \in Z$ if some vertex of A is adjacent to v. Since every connected component in D is a tree and has only one edge to v, some vertex in it has to have a neighbour in Z, otherwise Reduction Rule 1 would apply. Since we have that $|Z| \leq 2(k+\ell)$ and every vertex in D is adjacent to some vertex in Z, we may apply Expansion Lemma with $q = \ell+2$. This means, that in polynomial time, we can compute a nonempty set $\widehat{Z} \subseteq Z$ and a set of connected components $\widehat{D} \subseteq D$ such that:

1. $N_G(\bigcup_{D \in \widehat{D}} D) = \widehat{Z} \cup \{v\}$, and
2. Each $z \in \widehat{Z}$ will have $\ell+2$ private components $A_z^1, A_z^2, \ldots A_z^{\ell+2} \in \widehat{D}$ such that $z \in N_G(A_z^i)$ for all $i \in [\ell+2]$. By private we mean that the components $A_z^1, A_z^2, \ldots A_z^{\ell+2}$ are all different for different $z \in \widehat{Z}$.

Lemma 7.18. *For any ℓ-forest deletion set X of G that does not contain v, there exists an ℓ-forest deletion set X' in G such that $|X'| \leq |X|$, $X' \cap (\bigcup_{A \in \widehat{D}} A) = \emptyset$ and $\widehat{Z} \subseteq X'$.*

Proof. Let X be an ℓ-forest deletion set of G. We take $X' = (X \setminus \bigcup_{A \in \widehat{D}} A) \cup \widehat{Z}$. First we prove that $|X'| \leq |X|$. By definition of X' we just need to show that $|X \cap (\widehat{Z} \cup \bigcup_{A \in \widehat{D}} A)| \geq |\widehat{Z}|$. For each $z \in \widehat{Z}$, let us look at the graph induced on $Y_z = (\bigcup_{i \in [\ell+2]} A_z^i) \cup \{v, z\}$. Since each of the connected components A_z^i are trees for all $i \in [\ell+2]$, there exists a unique path from v to z through each A_z^i. Hence, there are $\ell+2$ vertex disjoint paths from v to z in

$G[Y_z]$. This means that for any edge set F of size at most ℓ, deleting F will not result in a forest (as two vertex disjoint paths will still remain, which would form a cycle) and hence $G[Y_z]$ is not an ℓ-forest. Let $Y = \bigcup_{z \in \widehat{Z}} Y_z$. In $G[Y]$, we have $|\widehat{Z}|$ graphs which are not ℓ forests and are vertex disjoint except for the vertex v. But we have assumed that $v \notin X$, so all these $|\widehat{Z}|$ graphs contribute at least 1 to X. Hence X contains at least $|\widehat{Z}|$ vertices from $\widehat{Z} \cup \bigcup_{A \in \widehat{D}} A$.

To show that X' is an ℓ-forest deletion set for G, let $F = \bigcup_{A \in \widehat{D}} A$. We look at the graph $G - (X' \cup F)$ and call it G'. Since G' is a subgraph of $G - X$, it is an ℓ-forest. Observe that G' is same as $(G - X') - F$. Since $G[F]$ is a forest and each of the trees in $G[F]$ is connected to $G - X'$ via a single edge (to v), there is no cycle passing through any vertex of F in $G - X'$. Hence, by Lemma 7.4, $(G - X')$ is an ℓ-forest and X' is an ℓ-forest deletion set for G. $\square$

Now we are ready to give the final reduction rule.

Reduction Rule 12. *Delete all edges between v and $\bigcup_{A \in \widehat{D}} A$ and put $\ell + 2$ parallel edges between v and z for all $z \in \widehat{Z}$.*

Lemma 7.19. *Reduction Rule 12 is correct.*

Proof. Let (G, k, ℓ) be a YES instance of ALMOST FOREST DELETION and let X be an ℓ-forest deletion set of size at most k for G. If $v \in X$, then X is an ℓ-forest deletion set for G' also because $G - \{v\}$ is same as $G' - \{v\}$. Now, if $v \notin X$, then we can assume by Lemma 8.30, that $\widehat{Z} \in X$, but then again, $G' - \widehat{Z}$ is a subgraph of $G - \widehat{Z}$ and X remains to be an ℓ-forest deletion set.

For the converse, let (G', k, ℓ) be a YES instance and X be an ℓ-forest deletion set of size at most k for G'. Again, if $v \in X$, then $G - X$ is same as $G' - X$, and hence G is a YES instance. Now, if $v \notin X$, then we know that $z \in X$ for all $z \in \widehat{Z}$, because we have added $\ell + 2$ parallel edges between v and z. So we have that $\widehat{Z} \in X$. Let $F = \bigcup_{A \in \widehat{D}} A$. Looking at $(G - (X \cup F))$, we know that it is a subgraph of $(G' - X)$ and hence is an ℓ-forest. Also, we observe that vertices of F do not take part in any cycle in $G - X$ and $(G - (X \cup F))$ is same as $((G - X) - F)$. But we know that $(G - (X \cup F))$ is an ℓ-forest and hence by lemma 7.4, $(G - X)$ is also an ℓ-forest. So we have that (G, k, ℓ) is a Yes instance of Almost Forest Deletion. □

Theorem 7.3. Almost Forest Deletion *admits a kernel with* $\mathcal{O}(k\ell(k + \ell))$ *vertices and* $\mathcal{O}(k\ell(k + \ell))$ *edges.*

Proof. Let us first see that we have a kernel when none of the reduction rules apply. If the maximum degree of the graph is at most $(k+\ell)(3\ell+8)$, and none of reduction rules 4-6 apply, then by Lemma 8.33 and Reduction Rule 7, we have a kernel with $\mathcal{O}(k\ell(k + \ell))$ vertices and $\mathcal{O}(k\ell(k + \ell))$ edges. Otherwise (if the maximum degree is greater than $(k + \ell)(3\ell + 8)$), one of reduction rules 8-12 applies.

Now we have to show that these reduction rules can be applied only polynomially many times. For that, we use the measure approach. We define a measure for the input graph which satisfies the following three properties.

1. It is polynomial in size of the graph initially.
2. It is always non-negative.
3. It decreases by a non-zero constant after application of each reduction rule, if the reduction rule does not terminate the algorithm.

Let $E_{\leq \ell+2} \subseteq E(G)$ denote the set of edges of G with multiplicity at most $\ell + 2$. We define the measure for the graph G to be the following.

$$\phi(G) = 2|V(G)| + |E_{\leq \ell+2}|$$

Clearly, the measure is polynomial in the size of the graph G initially and remains non negative throughout. Reduction rules 4, 8, 9 and 10 delete some vertex from the graph, and hence decrease $|V(G)|$ while not increasing $|E_{\leq \ell+2}|$. Reduction Rule 6, when applicable, reduces $|E_{\leq \ell+2}|$ by at least 1 while not changing $|V(G)|$. Reduction rules 7 and 11 terminate the algorithm when applicable. Hence, the only reduction rules which remain to be examined are 5 and 12. In Reduction Rule 5, if the degree two vertex being deleted has two distinct neighbours, then we decrease $|V(G)|$ and $|E_{\leq \ell+2}|$ by at least 1, as we lose 2 edges and gain at most 1. Now, if both the edges are adjacent to one vertex, then we might be increasing $|E_{\leq \ell+2}|$ by 1: by putting a loop on a vertex which did not have any loop. But because of a multiplicative factor of 2 for $|V(G)|$, the measure $\phi(G)$ drops by at least 1. While applying Reduction Rule 12, we delete a nonzero number of edges of multiplicity 1, and hence $|E_{\leq \ell+2}|$ and $\phi(G)$ decrease by at least 1. This concludes the proof of the theorem. $\square$

7.5 An $O^*(c^{\mathbf{tw}})$ algorithm for Almost Forest Deletion

In this section, we first design an algorithm, which given an instance (G, k, ℓ) of Almost Forest Deletion along with a tree decomposition of G of width at most $\mathbf{tw}$, solves it in time $O^*(c^{\mathbf{tw}})$. Then, using that algorithm, we give a subexponential algorithm for Almost Forest Deletion on planar graphs.

Before describing the algorithm, we will need the notions of Matroids and their representative families. For a broader overview on Matroids we refer to [Oxl10].

7.5.1 Matroids and Representative Family

Definition 7.4. *A pair $M = (E, \mathcal{I})$, where E is a ground set and $\mathcal{I}$ is a family of subsets (called independent sets) of E, is a* matroid *if it satisfies the following conditions:*

(I1) $\phi \in \mathcal{I}$.

(I2) *If $A' \subseteq A$ and $A \in \mathcal{I}$ then $A' \in \mathcal{I}$.*

(I3) *If $A, B \in \mathcal{I}$ and $|A| < |B|$, then $\exists\ e \in (B \setminus A)$ such that $A \cup \{e\} \in \mathcal{I}$.*

The axiom (I2) is also called the hereditary property and a pair $(E, \mathcal{I})$ satisfying only (I2) is called hereditary family. An inclusion wise maximal set of $\mathcal{I}$ is called a *basis* of the matroid. Using axiom (I3) it is easy to show that all the bases of a matroid have the same size. This size is called the *rank* of the matroid M, and is denoted by $\mathsf{rank}(M)$. The *uniform matroids* are among the simplest examples of matroids. A pair $M = (E, \mathcal{I})$ over an n-element ground set E, is called a uniform matroid if the family of independent sets is given by $\mathcal{I} = \{A \subseteq E \mid |A| \leq k\}$, where k is some constant. This matroid is also denoted as $U_{n,k}$.

7.5.2 Linear Matroids and Representable Matroids

Let A be a matrix over an arbitrary field $\mathbb{F}$ and let E be the set of columns of A. Given A we define the matroid $M = (E, \mathcal{I})$ as follows. A set $X \subseteq E$ is independent (that is $X \in \mathcal{I}$) if the corresponding columns are linearly independent over $\mathbb{F}$. The matroids that can be defined by such a construction are called *linear matroids*, and if a matroid can be defined by a matrix A over a field $\mathbb{F}$, then we say that the matroid is representable over $\mathbb{F}$. That is, a matroid $M = (E, \mathcal{I})$ of rank d is representable over a field $\mathbb{F}$ if there exist vectors in $\mathbb{F}^d$ correspond to the elements such that linearly independent sets of vectors correspond to independent sets of the matroid. A matroid $M = (E, \mathcal{I})$ is called *representable* or *linear* if it is representable over some field $\mathbb{F}$.

7.5.3 Graphic Matroids

Given a graph G, a graphic matroid $M = (E, \mathcal{I})$ is defined by taking elements as edges of G (that is $E = E(G)$) and $F \subseteq E(G)$ is in $\mathcal{I}$ if it forms a spanning forest in the graph G. The graphic matroid is representable over any field of size at least 2. Consider the matrix A_M with a row for each vertex $i \in V(G)$ and a column for each edge $e = ij \in E(G)$. In the column corresponding to $e = ij$, all entries are 0, except for a 1 in i or j (arbitrarily) and a -1 in the other. This is a representation over reals. To obtain a representation over a field $\mathbb{F}$, one simply needs to take the representation given above over reals and simply replace all -1 by the additive inverse of 1 .

Proposition 7.20 ([Oxl10]). *Graphic matroids are representable over any field of size at least* 2.

7.5.4 Representative Family

In this section we define q-representative family of a given family and state Theorems [FLS14] regarding its computation.

Definition 7.5 (**q-Representative Family** [FLS14])**.** *Given a matroid $M = (E, \mathcal{I})$ and a family $\mathcal{S}$ of subsets of E, we say that a subfamily $\widehat{\mathcal{S}} \subseteq \mathcal{S}$ is* q-representative *for $\mathcal{S}$ if the following holds: for every set $Y \subseteq E$ of size at most q, if there is a set $X \in \mathcal{S}$ disjoint from Y with $X \cup Y \in \mathcal{I}$, then there is a set $\widehat{X} \in \widehat{\mathcal{S}}$ disjoint from Y with $\widehat{X} \cup Y \in \mathcal{I}$. If $\widehat{\mathcal{S}} \subseteq \mathcal{S}$ is q-representative for $\mathcal{S}$ we write $\widehat{\mathcal{S}} \subseteq^q_{rep} \mathcal{S}$.*

In other words if some independent set in $\mathcal{S}$ can be extended to a larger independent set by q new elements, then there is a set in $\widehat{\mathcal{S}}$ that can be extended by the same q elements. A weighted variant of q-representative families is defined as follows. It is useful for solving problems where we are looking for objects of maximum or minimum weight.

Definition 7.6 (**Max q-Representative Family** [FLS14])**.** *Given a matroid $M = (E, \mathcal{I})$, a family $\mathcal{S}$ of subsets of E and a non-negative weight function $w : \mathcal{S} \to \mathbb{N}$ we say that a subfamily $\widehat{\mathcal{S}} \subseteq \mathcal{S}$ is* max q-representative *for $\mathcal{S}$ if the following holds: for every set $Y \subseteq E$ of size at most q, if there is a set $X \in \mathcal{S}$ disjoint from Y with $X \cup Y \in \mathcal{I}$, then there is a set $\widehat{X} \in \widehat{\mathcal{S}}$ disjoint from Y with*

1. *$\widehat{X} \cup Y \in \mathcal{I}$; and*

2. *$w(\widehat{X}) \geq w(X)$.*

We use $\widehat{\mathcal{S}} \subseteq^q_{maxrep} \mathcal{S}$ to denote a max q-representative family for $\mathcal{S}$.

We say that a family $\mathcal{S} = \{S_1, \ldots, S_t\}$ of independent sets is a *p-family* if each set in $\mathcal{S}$ is of size p. We state three lemmata providing basic results about representative family. These lemmata works for weighted variant representative family.

Lemma 7.21 ([FLS14])**.** *Let $M = (E, \mathcal{I})$ be a matroid and $\mathcal{S}$ be a family of subsets of E. If $\mathcal{S}' \subseteq^q_{rep} \mathcal{S}$ and $\widehat{\mathcal{S}} \subseteq^q_{rep} \mathcal{S}'$, then $\widehat{\mathcal{S}} \subseteq^q_{rep} \mathcal{S}$.*

Lemma 7.22 ([FLS14]). *Let $M = (E, \mathcal{I})$ be a matroid and $\mathcal{S}$ be a family of subsets of E. If $\mathcal{S} = \mathcal{S}_1 \cup \cdots \cup \mathcal{S}_\ell$ and $\widehat{\mathcal{S}}_i \subseteq^q_{rep} \mathcal{S}_i$, then $\cup_{i=1}^{\ell} \widehat{\mathcal{S}}_i \subseteq^q_{rep} \mathcal{S}$.*

Theorem 7.7 ([FLS14]). *Let $M = (E, \mathcal{I})$ be a linear matroid of rank $p + q = k$, $\mathcal{S} = \{S_1, \ldots, S_t\}$ be a p-family of independent sets and $w : \mathcal{S} \to \mathbb{N}$ be a non-negative weight function. Then there exists $\widehat{\mathcal{S}} \subseteq^q_{maxrep} \mathcal{S}$ of size $\binom{p+q}{p}$. Moreover, given a representation A_M of M over a field $\mathbb{F}$, we can find $\widehat{\mathcal{S}} \subseteq^q_{maxrep} \mathcal{S}$ of size at most $\binom{p+q}{p}$ in $\mathcal{O}\left(\binom{p+q}{p} tp^\omega + t\binom{p+q}{q}^{\omega-1}\right)$ operations over $\mathbb{F}$, where ω denotes the matrix multiplication exponent.*

Now we are ready to present the $O^*(c^{\mathbf{tw}})$ algorithm for Almost Forest Deletion. Instead of looking for an ℓ-forest deletion set of minimum size, we will look for an ℓ-forest of maximum size. That is, instead of saying an ℓ-forest deletion set Y to be the solution, in this section, we will say that $G - Y$ is a solution. We call $V' \subseteq V(G)$ an optimal solution if V' is a solution (an ℓ-forest) with maximum number of vertices. Let $\mathscr{S}$ be the set of vertex subsets such that for all $L \in \mathscr{S}$, L corresponds to an optimal solution. Let $(\mathcal{T}, f)$ be a tree decomposition of G with width $\mathbf{tw}$. For each tree node t, an integer $\ell' \in [\ell] \cup \{0\}$ and $Z \subseteq X_t$, we define a family of partial solutions, $S_t[\ell', Z]$ as follows.

$$S_t[\ell', Z] = \{U \subseteq V(G_t) \mid U \cap X_t = Z \text{ and } \mathsf{ex}(G_t[U]) = \ell'\}$$

We take $S_t[Z] = \cup_{\ell'=0}^{\ell} S_t[\ell', Z]$. We denote by K^t a complete graph on the vertex set X_t. Let G^* be subgraph of G. Let $C_1', \ldots, C_\ell'$ be the connected components of G^* that have nonempty intersection with X_t. Let $C_i = C_i' \cap X_t$. By $F(G^*)$ we denote the forest $\{Q_1, \ldots, Q_\ell\}$ where each Q_i is an arbitrary spanning tree of $K^t[C_i]$.

For two family of vertex subsets $\mathcal{P}$ and $\mathcal{Q}$ of a graph G, we denote

$$\mathcal{P} \otimes \mathcal{Q} = \{U_1 \cup U_2 \mid U_1 \in \mathcal{P}, U_2 \in \mathcal{Q} \text{ and } G_t[U_1 \cup U_2] \text{ is an } \ell\text{-forest } \}.$$

For every node t of $\mathcal{T}$, and $Z \subseteq X_t$, we store a family $\widehat{\mathcal{S}}_t[Z]$ of subsets of vertices of G_t satisfying the following.

Correctness Invariant: For every $L \in \mathscr{S}$ we have the following. Let $L_t = V(G_t) \cap L$, $L_R = L \setminus L_t$ and $L \cap X_t = Z$. Then there exists $\hat{L}_t \in \widehat{\mathcal{S}}_t[Z]$ such that $\hat{L} = \hat{L}_t \cup L_R$ is an optimal solution, i.e $G[\hat{L}_t \cup L_R]$ is an ℓ-forest with $|\hat{L}_t| \geq |L_t|$. Thus we have that $\hat{L} \in \mathscr{S}$.

We do dynamic programming on the tree decomposition in the bottom up manner, ensuring that the correctness invariant mentioned above is satisfied. To get the desired running time, we make use of representative sets. By using that, we will obtain $\widehat{\mathcal{S}}_t'[Z] \subseteq \widehat{\mathcal{S}}_t[Z]$ of small size. By small here, we mean that for a node t in the tree and for all $Z \subseteq X_t$, the size of $\widehat{\mathcal{S}}_t'[Z]$ is bounded by $\ell \cdot 2^{|Z|}$. We state the size requirements more precisely in the size invariant below.

Size Invariant: After node t of $\mathbb{T}$ is processed by the algorithm, we have that $|\widehat{\mathcal{S}}_t[\ell', Z, i]| \leq \binom{|Z|}{i}$, where $\widehat{\mathcal{S}}_t[\ell', Z, i]$ is the set of partial solutions in $\widehat{\mathcal{S}}_t[\ell', Z]$ that have i connected components with nonempty intersection with X_t.

Lemma 7.23. *Let t be a join node of $\mathbb{T}$ with children t_1 and t_2. Let $Z \subseteq X_t$ be a set of size k. Let $\widehat{\mathcal{S}}_{t_1}[Z]$ and $\widehat{\mathcal{S}}_{t_2}[Z]$ be two families of vertex subsets of $V(G_{t_1})$ and $V(G_{t_2})$ satisfying the size and correctness invariants. Furthermore, let $\widehat{\mathcal{S}}_t[Z] = \widehat{\mathcal{S}}_{t_1}[Z] \otimes \widehat{\mathcal{S}}_{t_2}[Z]$ be the family of vertex subsets of $V(G_t)$ satisfying the correctness invariant. Then in time $16^k n^{\mathcal{O}(1)}$ we can compute $\widehat{\mathcal{S}}_t'[Z] \subseteq \widehat{\mathcal{S}}_t[Z]$ satisfying correctness and size invariants.*

Proof. We start by associating a matroid with node t and the set $Z \subseteq X_t$ as follows. We consider a graphic matroid $M = (E, \mathcal{I})$ on $K^t[Z]$. Here, the element set E of the matroid is the edge set $E(K^t[Z])$ and the family of independent sets $\mathcal{I}$ consists of spanning forests of $K^t[Z]$. Here our objective is to find a small subfamily of $\widehat{\mathcal{S}}_t[Z] = \widehat{\mathcal{S}}_{t_1}[Z] \otimes \widehat{\mathcal{S}}_{t_2}[Z]$ satisfying correctness and size invariants using efficient computation of representative family in the graphic matroid M. For a set $U \in \widehat{\mathcal{S}}_t[Z]$, it is natural to associate $F(G[U])$ as the corresponding independent set in the graphic matroid.

Given $\widehat{\mathcal{S}}_{t_1}[Z]$ and $\widehat{\mathcal{S}}_{t_2}[Z]$, we first compute the set $\widehat{\mathcal{S}}_t[Z] = \widehat{\mathcal{S}}_{t_1}[Z] \otimes \widehat{\mathcal{S}}_{t_2}[Z]$. Since $\widehat{\mathcal{S}}_{t_1}[Z]$ and $\widehat{\mathcal{S}}_{t_2}[Z]$ satisfy the size invariant, we have that $|\widehat{\mathcal{S}}_t[Z]| \leq \ell^2 4^k$. Then we partition $\widehat{\mathcal{S}}_t[Z]$ into sets $\widehat{\mathcal{S}}_t[\ell', Z]$ for all $\ell' \in [\ell] \cup \{0\}$ by ensuring that if for $A \in \widehat{\mathcal{S}}_t[Z]$, $\mathsf{ex}(G[A]) = \ell'$, then

$A \in \widehat{\mathcal{S}}_t[\ell', Z]$. Let $\widehat{\mathcal{S}}_t[\ell', Z] = \{A_1, \ldots, A_p\}$ for some ℓ'. Let $\mathcal{L} = \{F(G[A_1]), \ldots, F(G[A_p])\}$ be the set of forests in $K^t[Z]$ corresponding to the vertex subsets in $\widehat{\mathcal{S}}_t[\ell', Z]$. For each $F(G[A_i]) \in \mathcal{L}$ we set $w(F(G[A_i])) = |A_i|$. For $i \in \{0, \ldots, k-1\}$, let $\mathcal{L}_i$ be the family of forests in $\mathcal{L}$ with i edges. Now we apply Theorem 7.7 and compute $\widehat{\mathcal{L}}_{i,j} \subseteq^{k-1-i-j}_{maxrep} \mathcal{L}_i$ for all j such that $i + j \leq k-1$, each of size $\binom{k-1}{i}$ in time $\mathcal{O}(\ell^2 4^k \binom{k}{i}^{w-1})$ (because $|\mathcal{L}_i| \leq \ell^2 \cdot 4^k$). Now we take $\widehat{\mathcal{L}}_i = \cup_j \widehat{\mathcal{L}}_{i,j}$. We have that $|\widehat{\mathcal{L}}_i| \leq k\binom{k-1}{i} \leq \binom{k}{i}$, and computing it takes $\mathcal{O}(k\ell^2 4^k \binom{k}{i}^{w-1})$ time. Let $\widehat{\mathcal{S}}'_t[\ell', Z, k-i] \subseteq \widehat{\mathcal{S}}_t[\ell', Z, k-i]$ be such that for every $A \in \widehat{\mathcal{L}}_i$, we choose exactly one $U \in \widehat{\mathcal{S}}_t[\ell', Z, k-i]$ to be in $\widehat{\mathcal{S}}'_t[\ell', Z, k-i]$ for which we have that $F(G[U]) = A$. Also, we know that $|\widehat{\mathcal{L}}_i| \leq \binom{k}{i}$ and hence $|\widehat{\mathcal{S}}'_t[\ell', Z, k-i]| \leq \binom{k}{i} = \binom{k}{k-i}$ as well. Hence, $\widehat{\mathcal{S}}'_t[Z]$ maintains the size invariant. To compute $\widehat{\mathcal{S}}_t[Z]$ and partitioning it into $\widehat{\mathcal{S}}_t[\ell', Z]$, it takes $\mathcal{O}(\ell^2 \cdot 4^k)$ time. To find $\widehat{\mathcal{L}}_j$ it takes time $\mathcal{O}(k\ell^2 4^k \binom{k}{j}^{w-1})$ for all $j \in [k]$. The total time to compute all the $\widehat{\mathcal{L}}_j$'s, and hence all the $\widehat{\mathcal{S}}'_t[\ell', Z, i]$'s, is $\sum_{j=0}^{k-1} \mathcal{O}(\ell^2 4^k \binom{k}{j}^{w-1}) \leq 16^k n^{\mathcal{O}(1)}$. Hence, $\widehat{\mathcal{S}}'_t[Z]$ can be computed in desired time and it satisfies the size invariant.

Now we show that the $\widehat{\mathcal{S}}'_t[Z]$ maintains the correctness invariant. Let $L \in \mathscr{S}$ and let $L_t = V(G_t) \cap L$, $L_R = L \setminus L_t$ and $Z = L \cap X_t$. Since $\widehat{\mathcal{S}}_t[Z]$ satisfies correctness invariant, there exists $\hat{L}_t \in \widehat{\mathcal{S}}_t[Z]$ such that $w(\hat{L}_t) \geq w(L_t)$, $\hat{L} = \hat{L}_t \cup L_R$ is an optimal solution and $\hat{L} \cap X_t = Z$. Let J be a spanning forest of $G[\hat{L}_t]$ and H be a spanning forest of $G[\hat{L}]$ such that $J \subseteq H$. Let $H_1 = H[\hat{L}_t]$ and $H_2 = H[\hat{L}_R \cup Z]$. We associate $F(H_1)$ and $F(H_2 \setminus E(Z))$ corresponding to H_1 and H_2 respectively in the graphic matroid M of $E(K^t[Z])$. For two forests $U_1 \subseteq G_t$ and $U_2 \subseteq G[(V \setminus V(G_t) \cup Z]$ such that $V(U_1) \cap X_t = V(U_2) \cap X_t = Z$, it is not hard to see that $U_1 \cup U_2$ is a forest in G if and only if $F(U_1) \cup F(U_2 \setminus E(Z))$ is a forest in $K^t[Z]$.

Let $\mathsf{ex}(G[\hat{L}_t]) = \ell'$, $|F(H_1)| = p$ and $|F(H_2 \setminus E(Z))| = q$. Since $H_1 \cup H_2$ forms a forest, so does $F(H_1) \cup F(H_2 \setminus E(Z))$ and hence $p + q \leq k-1$. Let $\mathcal{L}_i$ be the set of forests in $K^t[Z]$ having i edges corresponding to partial solutions whose excess is exactly ℓ'. We know that $F(H_1) \in \mathcal{L}_p$. Since we have computed $\widehat{\mathcal{L}}_{i,j} \subseteq^{k-1-i-j}_{maxrep} \mathcal{L}_i$ for all i, j such that $i + j \leq k-1$, we know that there exists $Y \in \widehat{\mathcal{L}}_{p,q}$ such that $w(Y) \geq w(F(H_1))$ and $Y \cup H_2$ forms a forest. So, there exists $\hat{L}'_t \in \widehat{\mathcal{S}}'_t[\ell', Z]$ such that $F(G[\hat{L}'_t]) = Y$, $\mathsf{ex}(G[\hat{L}'_t]) = \ell'$ and $|\hat{L}'_t| \geq \hat{L}_t$. Let J' be a spanning forest of $G[\hat{L}'_t]$. Clearly, $F(J') = F(G[\hat{L}'_t]) = Y$. Hence,

$J' \cup H_2$ is a forest on vertex set $\hat{L}'_t \cup L_R$. Let us call this vertex set $\hat{L}'$. Since $|\hat{L}'| \geq |\hat{L}|$, all we need to show is that $G[\hat{L}']$ is an ℓ-forest. Since H is a spanning forest of $G[\hat{L}]$, every edge in $G[L_R \cup Z]$ which is not in H_2 contributes at least 1 to $\mathsf{ex}(G[\hat{L}'])$. Let the number of such edges be c. Hence, we have have that $\mathsf{ex}(G[\hat{L}]) = \ell' + c$, but since we also have that $\hat{L}$ is an ℓ-forest, this gives $\ell' + c \leq \ell$. Since $J' \cup H_2$ is a forest on vertex set $\hat{L}'$ and $|E(G[\hat{L}'])| \leq E(J' \cup H_2) + \ell' + c$, we have that $G[\hat{L}']$ is an ℓ-forest as well. This completes the proof of the lemma. □

Theorem 7.8. *Given an instance* (G, k) *of* ALMOST FOREST DELETION *along with its tree decomposition of width at most* $\mathbf{tw}$*, it can be solved in* $\mathcal{O}^*(c^{\mathbf{tw}})$ *time.*

Proof. We first explain the dynamic programming algorithm over the tree-decomposition $(\mathbb{T}, \mathcal{X})$ of G and prove that it maintains the correctness invariant. We assume that $(\mathbb{T}, \mathcal{X})$ is a nice tree-decomposition of G. By $\widehat{\mathcal{S}}_t$ we denote $\cup_{Z \subseteq X_t} \widehat{\mathcal{S}}_t[Z]$ (also called a *representative family of partial solutions*). We show how $\widehat{\mathcal{S}}_t$ is obtained by doing dynamic programming from base node to the root node.

Base node t. Here the graph G_t is empty and thus we take $\widehat{\mathcal{S}}_t = \emptyset$.

Introduce node t with child t'. Here, we know that $X_t \supset X_{t'}$ and $|X_t| = |X_{t'}| + 1$. Let v be the vertex in $X_t \setminus X_{t'}$. The graph $G_t = G_{t'} \setminus \{v\}$. So each partial solution in $G_{t'}$ is a partial solution in G_t or it differs at vertex v from a partial solution in G_t, i.e, for $i \in [\ell] \cup \{0\}$,

$$\widehat{\mathcal{S}}_t[i, Z] = \begin{cases} \widehat{\mathcal{S}}_{t'}[Z] & \text{if} \quad v \notin Z \\ \Big\{U \cup \{v\} \mid U \in \widehat{\mathcal{S}}_{t'}[Z \setminus \{v\}] \text{ and } \mathsf{ex}(G[U \cup \{v\}]) = i\Big\} & \text{if} \quad v \in Z \end{cases}$$

When $v \notin Z$, $\widehat{\mathcal{S}}_t[Z]$ satisfies correctness and size invariant. When $v \in Z$, $|\widehat{\mathcal{S}}_t[Z, i]| \leq 2^k$ and we can apply Theorem 7.7 by associating a family of independent sets in $K^t[Z]$ (like in Lemma 7.23) and find $\widehat{\mathcal{S}}'_t[i, Z, j] \subseteq \widehat{\mathcal{S}}_t[i, Z, j]$ satisfying correctness and size invariant in time $\mathcal{O}(2^k \binom{k}{i}^{w-1})$.

Forget node t with child t'. Here we know $X_t \subset X_{t'}$, $|X_t| = |X_{t'}| - 1$ and $G_t = G_{t'}$. Let $X'_t \setminus X_t = \{v\}$. So for any $Z \subseteq X_t$ we have $\widehat{\mathcal{S}}_t[Z] = \widehat{\mathcal{S}}_{t'}[Z] \cup \widehat{\mathcal{S}}_{t'}[Z \cup \{v\}]$. The number of elements in $\widehat{\mathcal{S}}_t[Z]$ with i number of connected components intersecting with X_t is upper bounded by $\binom{k+1}{i} + \binom{k+1}{i+1} \leq \binom{k+2}{i}$. Again by applying Theorem 7.7 we can find $\widehat{\mathcal{S}}'_t[i,Z,j] \subseteq \widehat{\mathcal{S}}_t[i,Z,j]$ satisfying correctness and size invariant in time $\mathcal{O}(\binom{k+2}{i}\binom{k}{i}^{w-1})$.

Join node t with two children t_1 and t_2. Here, we know that $X_t = X_{t_1} = X_{t_2}$. The natural way to get a family of partial solutions for X_t is the union of vertex sets of two families stored at node t_1 and t_2 which form an ℓ-forest, i.e,

$$\begin{aligned}\widehat{\mathcal{S}}_t[Z] &= \{U_1 \cup U_2 \mid U_1 \in \widehat{\mathcal{S}}_{t_1}[Z], U_2 \in \widehat{\mathcal{S}}_{t_2}[Z], G[U_1 \cup U_2] \text{ is an } \ell\text{-forest}\} \\ &= \widehat{\mathcal{S}}_{t_1}[Z] \otimes \widehat{\mathcal{S}}_{t_2}[Z]\end{aligned}$$

Now we show that $\widehat{\mathcal{S}}_t$ maintains the correctness invariant. Let $L \in \mathscr{S}$. Let $L_t = V(G_t) \cap L, L_{t_1} = V(G_{t_1}) \cap L, L_{t_2} = V(G_{t_2}) \cap L$ and $L_R = L \setminus L_t$. Let $Z = L \cap X_t$ Now observe that

$$\begin{aligned} L \in \mathscr{S} &\iff L_{t_1} \cup L_{t_2} \cup L_R \in \mathscr{S} \\ &\iff \hat{L}_{t_1} \cup L_{t_2} \cup L_R \in \mathscr{S} \;\; (\text{by the property of } \widehat{\mathcal{S}}_{t_1} \text{ we have } \hat{L}_{t_1} \in \widehat{\mathcal{S}}_{t_1}[Z]) \\ &\iff \hat{L}_{t_1} \cup \hat{L}_{t_2} \cup L_R \in \mathscr{S} \;\; (\text{by the property of } \widehat{\mathcal{S}}_{t_2} \text{ we have } \hat{L}_{t_2} \in \widehat{\mathcal{S}}_{t_2}[Z]) \end{aligned}$$

We put $\hat{L}_t = \hat{L}_{t_1} \cup \hat{L}_{t_2}$. By the definition of $\widehat{\mathcal{S}}_t[Z]$, we have that $\hat{L}_{t_1} \cup \hat{L}_{t_2} \in \widehat{\mathcal{S}}_t[Z]$. The above inequalities also show that $\hat{L} = \hat{L}_t \cup L_R \in \mathscr{S}$. Note that $(\hat{L}_t \cup L_R) \cap X_t = Z$ This concludes the proof of correctness invariant.

We apply Lemma 7.23 and find $\widehat{\mathcal{S}}'_t[Z] \subseteq \widehat{\mathcal{S}}_t[Z]$ satisfying correctness and size invariant in $16^k n^{\mathcal{O}(1)}$ time where $k = |Z|$.

Root node r. Here, $X_r = \emptyset$. Let (G, x, ℓ) was the input instance for Almost Forest Deletion. We go through all the solutions in $\widehat{\mathcal{S}}_r[\emptyset]$ and output Yes if there exists $L \in \widehat{\mathcal{S}}_r[\emptyset]$ such that $|L| \geq |V(G| - x$, No otherwise.

In worst case, in every tree node t, for all subset $Z \subseteq X_t$ such that $|Z| = k$, we apply Lemma 7.23 which takes $16^k n^{\mathcal{O}(1)}$ time. Let $|X_t| = t$. Then, total time taken by the algoritm to compute $\widehat{\mathcal{S}}_t[Z]$ for all $Z \subseteq X_t$ is $\sum_{i=0}^{|t|} \binom{t}{i} 16^i n^{\mathcal{O}(1)} = 17^t n^{\mathcal{O}(1)}$. Since the given tree decomposition $(\mathbb{T}, \mathcal{X})$ is of depth at most $\mathbf{tw}$, we have that $t \leq \mathbf{tw} + 1$ for all tree nodes t. Hence, for each tree node t, computing $\widehat{\mathcal{S}}_t[Z]$ for all $Z \subseteq X_t$ takes time $17^{\mathbf{tw}} n^{\mathcal{O}(1)}$. Hence, we have $17^{\mathbf{tw}} n^{\mathcal{O}(1)}$ as the final running time of the algorithm as the number of tree nodes in a nice tree decomposition are bounded by $\mathcal{O}(n)$. □

Now we state the following result. Here $\boxplus_t$ denotes a grid of dimension $t \times t$.

Theorem 7.9 (Planar Extended Grid Theorem [GT12, RST94]). *Let t be a nonnegative integer. Then every planar graph G of treewidth at least $\frac{9}{2}t$ contains $\boxplus_t$ as a minor. Furthermore, for every $\epsilon > 0$ there exists an $\mathcal{O}(n^2)$ algorithm that, for a given n-vertex planar graph G and integer t, either outputs a tree decomposition of G of width at most $(\frac{9}{2} + \epsilon)t$, or returns that $\boxplus_t$ is a minor of G.*

Now we show the following lemma, which related the size of the grid with size of its ℓ-forest deletion set.

Lemma 7.24. *Let X be an ℓ-forest deletion set of $\boxplus_t$ of size at most k, then $t \leq \sqrt{\ell + 3k} + 1$.*

Proof. Let $G = \boxplus_t$. It has t^2 vertices and $2t(t-1)$ edges. Hence, $\mathsf{ex}(G) = t^2 - 2t + 1 = (t-1)^2$. Also, since $G - X$ is an ℓ-forest, $\mathsf{ex}(G - X) \leq \ell$. We also know that degree of every vertex in G is at most 4, hence deleting it can decrease the excess by at most 3 (since the graph loses 1 vertex and at most 4 edges). So we get $\mathsf{ex}(G - X) \geq \mathsf{ex}(G) - 3|X|$. Putting the inequalities together, we get $\ell \geq (t-1)^2 + 3k$ or $t \leq \sqrt{\ell + 3k} + 1$, as desired. □

It is easy to see that the size of the ℓ-forest deletion set does not increase while performing any of the minor operations. Given an instance (G, k, ℓ) of Almost Forest Deletion, we invoke Theorem 7.9 on it with $t = \lceil \sqrt{\ell + 3k} \rceil + 1$. If the algorithm returns that $\boxplus_t$ is a minor of G then we can return that the instance is a No instance. Otherwise we get a tree decomposition of G of width at most $\mathcal{O}(\sqrt{\ell + k})$. Then we apply Theorem 7.8 to solve the problem in $2^{\mathcal{O}(\sqrt{\ell+k})} n^{\mathcal{O}(1)}$ time and get the following. x

Theorem 7.10. Almost Forest Deletion *can be solved in $2^{\mathcal{O}(\sqrt{\ell+k})} n^{\mathcal{O}(1)}$ time on planar graphs.*

7.6 Conclusions

In this chapter we studied Almost Forest Deletion and obtained a polynomial kernel as well as a single exponential time algorithm for the problem. It would be interesting to study other classical problems from this view-point of distance from tractability using a suitable measure of distance.

Chapter 8

Parameterized Algorithm and Uniform Kernel for Pseudoforest Deletion

8.1 Introduction

The main goal of this chapter is to study the following interesting problem: How can we generalize the family of forests such that the nice structural properties of forests and the interesting algorithmic properties of FVS can be extended to problems on this class? There are two ways of quantitatively generalizing forests: given a positive integer ℓ we define graph classes $\mathcal{G}_\ell$ and $\mathcal{F}_\ell$. The graph class $\mathcal{G}_\ell$ is defined as those graphs that can be transformed into a forest by deleting at most ℓ edges. On the other hand the graph class, $\mathcal{F}_\ell$ contains all graphs where each connected component can be transformed into a forest by deleting at most ℓ edges. Graphs in $\mathcal{G}_\ell$ are called *almost ℓ-forest* (see Chapter 7). The class $\mathcal{F}_1$ is known as pseudoforest in the literature and we call $\mathcal{F}_\ell$ as *ℓ-pseudoforest.* In this chapter we study the problem of deleting k-vertices to get into $\mathcal{F}_\ell$, ℓ-PSEUDOFOREST DELETION, in the realm of parameterized complexity.

In chapter 7, we looked at a generalization of FVS in terms of $\mathcal{G}_\ell$. In particular we studied the problem of deleting k-vertices to get into $\mathcal{G}_\ell$, which we called Almost Forest Deletion, parameterized by k and ℓ and obtained an algorithm with with running time $2^{\mathcal{O}(k+\ell)}n^{\mathcal{O}(1)}$ and a kernel of size $\mathcal{O}(k\ell(k+\ell))$. One property of almost-ℓ-forests which is crucial in the design of FPT and kernelization algorithms of the previous chapter was that any almost-ℓ-forests on n vertices can have at most $n+\ell-1$ edges. The same can not be said about ℓ-pseudoforests and they can turn out to be significantly more dense. So while the techniques used for arriving at FPT and kernelization results for FVS give similar results for Almost Forest Deletion, they break down when applied directly to ℓ-pseudoforest Deletion. So we had to get into the theory of *protrusions.* Protrusions of a graph are subgraphs which have small boundary and a small treewidth. A protrusion replacer is an algorithm which identifies large protrusions and replaces them with smaller ones. Fomin et al. [FLMS12] use protrusion-replacer to arrive at FPT and kernelization results for Planar-$\mathcal{F}$ Deletion.

We first apply the techniques used in [FLMS12] to get an FPT algorithm for ℓ-pseudoforest Deletion. To that end, we have to show that ℓ-pseudoforest Deletion has a protrusion replacer, which we do by showing that the property of being an ℓ-pseudoforest is strongly monotone and minor-closed. We arrive at a running time of $\mathcal{O}^*(c_\ell^k)$ for ℓ-pseudoforest Deletion where c_ℓ is a function of ℓ alone. If we try to apply the machinery of [FLMS12] to get a kernelization algorithm for ℓ-pseudoforest Deletion, it only gives a kernel of size k^c where the constant c depends on ℓ. We use the similarity of ℓ-pseudoforest Deletion with FVS and apply Gallai's theorem and Expansion Lemma to decrease the maximum degree of the graph. This, when combined with techniques used in [FLMS12], gives us a kernel of size ck^2, where the constant c depends on ℓ. These kind of kernels are more desired as it gives $\mathcal{O}(k^2)$ kernel for every fixed ℓ, while the non-uniform kernelization does give a polynomial kernel for every fixed ℓ, but the exponent's dependency on ℓ makes the size of the kernel grow very quickly when compared to uniform-kernelization case. This result is one of the main results of the chapter and should be viewed as another result similar to the one about hitting forbidden minors obtained recently by Giannopoulou et al. [GJLS15].

We also look at a special case for of ℓ-pseudoforest Deletion, namely Pseudoforest

Deletion, where we ask whether we can delete at most k vertices to get to a *pseudoforest.* A pseudoforest is special case of ℓ-pseudoforest for $\ell = 1$, i.e. in a pseudoforest, each connected component is just one edge away from being a tree. In other words, it is the class of graphs where every connected component has at most one cycle. We apply the well known technique of iterative compression along with a non-trivial measure and an interesting base case to arrive at an $\mathcal{O}^*(7.5618^k)$ algorithm for this problem. We also give an explicit kernel with $\mathcal{O}(k^2)$ vertices for the problem.

8.2 Preliminaries

In this section, we first give the notations and definitions which are used in this chapter. Then we state some basic properties about ℓ-pseudoforests and some known results which will be used.

Notations and Definitions: For $0 < \alpha \leq 1$, we say that a vertex subset $S \subseteq V(G)$ is an α-cover of G, if the sum of vertex degrees $\sum_{v \in S} d(v)$ is at least $2\alpha|E(G)|$. A *forest* is a graph which does not contain any cycles. An *ℓ-pseudoforest* is a graph which every component is at most ℓ edges away from being a tree, i.e. the graph can be transformed into a forest by deleting at most ℓ edges from each of its connected components. When ℓ is equal to 1, we call the graph a *pseudoforest* instead of a 1-pseudoforest. For a connected component C of a graph, we call the quantity $|E(G[C])| - |C| + 1$ the *excess of* C and denote it by $\mathsf{ex}(C)$. It can also be equivalently defined as the minimum number of edges we need to delete from a connected component to get to a tree. For a graph G, let $\mathcal{C}$ be the set of its connected components. We define the *excess* of a graph G, denoted by $\mathsf{ex}(G)$ as follows.

$$\mathsf{ex}(G) = \mathsf{max}_{C \in \mathcal{C}} \mathsf{ex}(C)$$

It is easy to see that a graph G is an ℓ-pseudoforest if and only if $\mathsf{ex}(G) \leq \ell$. We denote by $\{1, \ldots, n\}$ by $[n]$. We define the ℓ-pseudoforest Deletion problem as follows.

ℓ-PSEUDOFOREST DELETION	**Parameter(s):** k
Input: A graph G, integers ℓ and k.	
Question: Does there exist $X \subseteq V(G)$ such that $G - X$ is an ℓ-pseudoforest?	

The set X is called an *ℓ-pseudoforest deletion set* of G. Similarly we can define PSEUDOFOREST DELETION to be the problem where we ask whether we can delete $S \subseteq V(G)$ such that $|S| \leq k$ and $G - S$ is a pseudoforest.

Minors. 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_{xy} 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. We say that a graph class $\mathcal{G}$ is minor closed if for all $G \in \mathcal{G}$, if $H \leq_m G$, then $H \in \mathcal{G}$. We say that a graph class $\mathcal{G}$ is characterized by $\mathcal{H}$ as forbidden minors if for all graphs G, $G \in \mathcal{G}$ if and only if for all $H \in \mathcal{H}$, $H \nleq_m G$. If $|\mathcal{H}|$ is finite, then we say that $\mathcal{G}$ has *finite forbidden minor characterization.*

t-Boundaried graphs and Gluing. A t-boundaried graph is a graph G and a set $B \subseteq V(G)$ of size at most t with each vertex $v \in B$ having a label $l_G(v) \in \{1, \dots, t\}$. Each vertex in B has a unique label. We refer to B as the boundary of G. For a t-boundaried G the function $\delta(G)$ returns the boundary of G. Observe that a t-boundaried graph may have no boundary at all.

Two t-boundaried graphs G_1 and G_2 can be glued together to form a graph $G = G_1 \oplus G_2$. The gluing operation takes the disjoint union of G_1 and G_2 and identifies the vertices of $\delta(G_1)$ and $\delta(G_2)$ with the same label. If there are vertices $u_1, v_1 \in \delta(G_1)$ and $u_2, v_2 \in \delta(G_2)$ such that $l_{G_1}(u_1) = l_{G_2}(u_2)$ and $l_{G_1}(v_1) = l_{G_2}(v_2)$ then G has vertices u formed by unifying u_1 and u_2 and v formed by unifying v_1 and v_2. The new vertices u and v are adjacent if $(u_1, v_1) \in E(G_1)$ or $(u_2, v_2) \in E(G_2)$.

Counting Monadic Second Order Logic (CMSO). The syntax of Monadic Second Order Logic (MSO) of graphs includes the logical connectives $\vee$, $\wedge$, $\neg$, $\Rightarrow$, $\Leftrightarrow$, variables for vertices, edges, sets of vertices, 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 sentences 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 the counting monadic second-order logic. Thus CMSO is MSO with the following atomic sentence for a set S:

$\mathbf{card}_{q,r}(S) = \mathbf{true}$ if and only if $|S| \equiv q \mod r$.

We now define a class of parameterized problems, called p-MIN-CMSO$[\psi]$ problems, with one problem for each CMSO sentence ψ on graphs, where ψ has a free vertex set variable S. The p-MIN-CMSO$[\psi]$ problem defined by ψ is denoted by p-MIN-CMSO$[\psi]$ and defined as follows.

p-MIN-CMSO$[\psi]$ **Parameter(s):** k
Input: A graph G and an integer k
Question: Is there a subset $S \subseteq V(G)$ such that $|S| \leq k$ and $(G, S) \models \psi$?

In other words, p-MIN-CMSO$[\psi]$ is a subset Π of $\Sigma^* \times \mathbb{N}$ where for every $(x, k) \in \Sigma^* \times \mathbb{N}$, $(x, k) \in \Pi$ if and only if there exists a set $S \subseteq V(G)$ where $|S| \leq k$ such that the graph G

encoded by x together with S satisfy ψ, i.e., $(G, S) \models \psi$. In this case, we say that Π is definable by the sentence ψ and that Π is a p-MIN-CMSO$[\psi]$.

Strong Monotonicity. By U_I we denote the set of all tuples (G, S) such that G is a boundaried graph having label set I and $S \subseteq V(G)$.

Let Π be a p-MIN-CMSO$[\psi]$ problem definable by some sentence ψ. We say that a tuple $(G', S') \in U_I$ is *ψ-feasible* for some boundaried graph G with label set I if there exist some $S \subseteq V(G)$ such that $(G \oplus G', S \cup S') \models \psi$. For a boundaried graph G with label set I, we define the function $\zeta_G : U_I \to \mathbb{Z}^+ \cup \{\infty\}$ as follows. For a structure $\alpha = (G', S') \in U_I$ we set

$$\zeta_G(\alpha) = \begin{cases} \mathsf{min}\{|S| \mid S \subseteq V(G) \wedge (G \oplus G', S \cup S') \models \psi\}, & \text{if } \alpha \text{ is } \psi\text{-feasible for } G, \\ \infty & \text{otherwise.} \end{cases}$$

A p-MIN-CMSO$[\psi]$ problem Π is *strongly monotone* if there exists a function $f : \mathbb{Z}^+ \to \mathbb{Z}^+$ such that the following condition is satisfied. For every boundaried graph G with label set I, there exists a subset $W \subseteq V(G)$ such that for every $(G', S') \in U_I$ such that $\zeta_G(G', S')$ is finite, it holds that $(G \oplus G', W \cup S') \models \psi$ and $|W| \leq \zeta_G(G', S') + f(|I|)$.

Protrusions and Protrusion Replacement. For a graph G and $S \subseteq V(G)$, we define $\partial_G(S)$ as the set of vertices in S that have a neighbour in $V(G) \setminus S$. An r-protrusion in a graph G is a set $X \subseteq V(G)$ such that $|\partial(X)| \leq r$ and $\mathbf{tw}(G[X]) \leq r$. Let G be a graph containing an r-protrusion X and let X' be an r-boundaried graph. Let $\hat{X} = X \setminus \partial(X)$. Then the act of *replacing* X by X' means replacing G by $\hat{G} \oplus X'$, where $\hat{G}$ is the boundaried graph $G - \hat{X}$ with boundary $\partial(X)$ where each vertex in $\partial(X)$ is assigned an arbitrary unique label from $\{1, \ldots, r\}$.

A protrusion replacer for a parameterized graph problem Π is a family of algorithms, with one algorithm for every constant r. The rth algorithm has the following specifications. There exists a constant r' (which depends on r) such that given an instance (G, k) and an r-protrusion X in G of size at least r', the algorithm runs in time $\mathcal{O}(|X|)$ and outputs an instance (G', k') such that $(G', k') \in \Pi$ if and only if $(G, k) \in \Pi$, $k' \leq k$ and G' is obtained from G by replacing X by an r-boundaried graph X' with less than r' vertices. Observe that since X has at least r' vertices and X' has less than r' vertices this implies that $|V(G')| < |V(G)|$.

Protrusion Decomposition. A graph G has an (α, β)-protrusion decomposition if $V(G)$ has a partition $P = R_0, R_1, \dots, R_t$ where

- $\mathsf{max}\{t, |R_0|\} \leq \alpha$,
- each $N_G[R_i]$ for $i \in [t]$ is a β-protrusion of G, and
- for all $i \geq 1$, $N(R_i) \subseteq R_0$.

We call the sets $R_i = N_G[R_i]$ for all $i \in [t]$ protrusions of P.

8.2.1 Preliminary results

In this subsection, we present some basic properties of ℓ-pseudoforests as well as some known results which will be used in other sections the chapter.

Observation 8.1. *Let G' be a subgraph of G. If G is an ℓ-pseudoforest, then so is G'.*

Observation 8.2. *Let G be an ℓ-pseudoforest and C be one of the connected components of G. Then $G[C]$ has at most $|C| - 1 + \ell$ edges.*

Lemma 8.3. *Let G be a graph and let $S \subseteq V(G)$ be such that for all $v \in S$, $d_G(v) \leq 1$. Then (G, k) is a* Yes *instance of ℓ-*pseudoforest Deletion *if and only if $(G - S, k)$ is a* Yes *instance.*

Proof. Let v be a vertex of degree at most 1 in G. It is sufficient to show that $(G - \{v\}, k)$ is a Yes instance of ℓ-pseudoforest Deletion if and only if (G, k) is a Yes instance. Then we can apply this iteratively to delete all the vertices in S to prove the lemma.

When $d(v) = 0$, then it easy to see that $\mathsf{ex}(G) = \mathsf{ex}(G')$. So, let us assume that $d(v) = 1$. Let $G' = (G - \{v\})$ and let C be the connected component of G which contains v. All the connected components of G except C remain unaffected by deletion of v and hence they are also connected components of G' and their excess remains the same. The only component for which excess could have changed is C. But C loses exactly one vertex and one edge and hence its excess also remains the same. So we have that $\mathsf{ex}(G) = \mathsf{ex}(G')$. This completes the proof of the lemma. □

Lemma 8.4. *Treewidth of an ℓ-pseudoforest is at most $\ell + 1$.*

Before we prove the lemma, we state two well known results about treewidth.

Lemma 8.5 (Folklore)**.** *Treewidth of a forest is* 1*.*

Lemma 8.6 (Folklore)**.** *Let $G' = G - S$ for some $S \subseteq V(G)$. Then $\mathbf{tw}(G) \leq \mathbf{tw}(G') + |S|$.*

Now we are ready to give proof of Lemma 8.4.

of Lemma 8.4. It easy to see that treewidth of the graph is maximum treewidth over all its components. Now we show that if G is an ℓ-pseudoforest, each connected component of G has treewidth at most $\ell + 1$. Let C be any connected component of G, which is also an ℓ-pseudoforest. Let $X \subseteq E(G[C])$ be the set of edges such that $(C, E(G[C]) \setminus X)$ is a forest and $|X| \leq \ell$. Let $S = \emptyset$ initially. For each edge $e \in X$, we pick an endpoint of e arbitrarily and include it in S. Clearly $|S| \leq \ell$ and $G[C \setminus S]$, being subgraph of $(C, E(G[C]) \setminus X)$, is a forest. Hence, by lemmas 8.5 and 8.6, we have that $\mathbf{tw}(C) \leq \ell + 1$. □

Theorem 8.1. *ℓ-*PSEUDOFOREST DELETION *has a protrusion replacer.*

For proving the theorem, we would like to make use of the following lemma.

Lemma 8.7 ([BFL+09])**.** *Every strongly monotone p-*MIN*-CMSO$[\psi]$ problem has a protrusion replacer.*

We show that ℓ-PSEUDOFOREST DELETION is a p-MIN-CMSO$[\psi]$ and is strongly monotone through a series of lemmata, which, when combined with Lemma 8.7, will give rise to proof of Theorem 8.1. Before we start that, we state a classical result by Robertson and Seymour.

Theorem 8.2 (Robertson and Seymour [RS04]). *If a graph class $\mathcal{G}$ is minor closed then it has finite forbidden minor characterization.*

Lemma 8.8. *The class of ℓ-pseudoforests has finite forbidden minor characterization.*

Proof. It is sufficient to show that the class of ℓ-pseudoforests is minor closed. We have already observed that subgraph of an ℓ-pseudoforest is also an ℓ-pseudoforest. So we just need to show that the edge contraction preserves the property of the graph being an ℓ-pseudoforest. Let G be an ℓ-pseudoforest and let $G' = G/e$. Let C be the connected component of G containing e and C' be corresponding connected component of G'. Since $G - C$ is same as $G' - C'$, we just need to show that $\mathsf{ex}(C) \leq \mathsf{ex}(C)$. It is indeed true because $|C'| = |C| - 1$ while $|E(G'[C'])| \leq |E(G[C])| - 1$. This completes the proof of the lemma. $\square$

Lemma 8.9. *There exists a CMSO statement η_H for a fixed graph H, which when instantiated with a graph G, evaluates to true if and only if G contains H as a minor.*

Proof. Given a graph H such that $V(H) = \{h_1, h_2, \ldots, h_c\}$, we define η_H to be following.

$$\eta_H := \exists X_1, X_2, \ldots X_c \subseteq V(G) \Big[\bigwedge_{i \neq j} (X_i \cap X_j = \emptyset) \wedge \bigwedge_{1 \leq i \leq c} \mathsf{Conn}(G, X_i) \wedge \\ \bigwedge_{(h_i, h_j) \in E(H)} \exists x \in X_i \wedge y \in X_j [(x, y) \in E(G)] \Big]$$

Where $\mathsf{Conn}(G, X)$ is instantiation of the standard CMSO connectivity statement ς with (G, X), which is such that $(H, Y) \models \varsigma$ if and only if $H[Y]$ is connected. The statement η_H says that there exist vertex disjoint connected subgraphs $G[X_1], G[X_2], \ldots G[X_c]$ of G such that whenever $(h_i, h_j) \in E(H)$, there exists at least one edge in G with endpoints in X_i and X_j. Clearly, the graph obtained by contracting all the edges inside the connected subgraphs $G[X_1], G[X_2], \ldots G[X_c]$ contains H as its subgraph. Hence we have that $H \leq_m G$. It is also easy to see from the definitions of minor, that if $H \leq_m G$ then $G \models \eta_H$. $\square$

Lemma 8.10. ℓ-PSEUDOFOREST DELETION *is a p-*MIN*-CMSO$[\psi]$.*

Proof. To show that ℓ-PSEUDOFOREST DELETION is a p-MIN-CMSO$[\psi]$, we need to show that there exists a CMSO sentence ψ such that for a graph G and $S \subseteq V(G)$, $(G, S) \models \psi$ if and only if S is an ℓ-pseudoforest deletion set of G. Suppose we can write the property of a graph being an ℓ-pseudoforest as a CMSO sentence ϕ, then we write the CMSO sentence ψ as following.

$$\psi(G, S) = \phi(G - S)$$

Where $\phi(G - S)$ is instantiation of ϕ with the graph $G - S$. To write ϕ, we know from Lemma 8.8 that the class of ℓ-pseudoforests has finite forbidden minor characterization. Let $\mathcal{H} = \{H_1, H_2, \ldots H_p\}$ be the set which characterizes the set of all ℓ-pseudoforests as forbidden minors. This means that a graph G is an ℓ-pseudoforest if and only if it does not contain any $H \in \mathcal{H}$ as a minor. We also know from Lemma 8.9, that for a fixed graph H, there exists a CMSO statement η_H, which when instantiated with a graph G, evaluates to true if and only if G contains H as a minor. So, we write ϕ as following.

$$\phi = (\neg\eta_{H_1}) \wedge (\neg\eta_{H_2}) \wedge \ldots \wedge (\neg\eta_{H_p})$$

which completes the description of desired sentence ψ. □

Lemma 8.11. ℓ-PSEUDOFOREST DELETION *is strongly monotone.*

Proof. For a given boundaried graph G with boundary I, let W^* be the ℓ-pseudoforest deletion set of G of minimum size. We show that ℓ-PSEUDOFOREST DELETION satisfies the criterion for being strongly monotone by putting $f(x) = 2x$ and the setting $W = W^* \cup I$. Now, it is sufficient to show that for every $(G', S') \in \mathcal{U}_I$ such that $\zeta_G(G', S')$ is finite, it holds that $(G \oplus G', W \cup S') \models \psi$ and $|W| \leq \zeta_G(G', S') + 2|I|$. Let $(G', S') \in \mathcal{U}_I$, and let S be the set of smallest cardinality such that $S \subseteq V(G)$ and $(G \oplus G', S \cup S') \models \psi$. Since $\zeta_G(G', S')$ is finite, such a set exists. Also, by definition of S, $\zeta_G(G', S') = |S|$. So now we just need to show that $(G \oplus G', W \cup S') \models \psi$ and $|W| \leq |S| + 2|I|$.

Let $G^* = G \oplus G'$ and $X = W \cup S'$. To show that $(G \oplus G', W \cup S') \models \psi$, we need to show that X is an ℓ-pseudoforest deletions set of G^*. Since $V(G) \cap V(G') = I$ and $I \subseteq X$, no connected component of $G^* - X$ contains vertices from both G and G'. Hence, $G^* - X$ is a subgraph of disjoint unions of $G - W$ and $G' - (S' \cup I)$, where $G - W$ is an ℓ-pseudoforest because it is a subgraph of $G - W^*$. For $G' - (S' \cup I)$, it is a subgraph of $G^* - (S \cup S')$ and since $(G \oplus G', S \cup S') \models \psi$, it is also an ℓ-pseudoforest.

To show that $|W| \leq |S| + 2|I|$, observe that since $(G \oplus G', S \cup S') \models \psi$ and $V(G) \cap V(G') = I$, we have that $(S \cup I)$ is an ℓ-pseudoforest deletion set of G. Hence, we have that $|S \cup I| \geq |W^*|$ or $|S| + |I| \geq |W| - |I|$ or $|W| \leq |S| + 2|I|$. □

Lemmas 8.10 and 8.11, when combined with Lemma 8.7, complete the proof of Theorem 8.1. Now we state the final result in the section.

Lemma 8.12 ([FLMS12]). *If an n-vertex graph G has a vertex subset X such that $\mathbf{tw}(G - X) \leq b$, then G admits a $((4|N[X]|)(b+1), 2(b+1))$-protrusion decomposition.*

8.3 A $c_\ell^k n^{\mathcal{O}(1)}$ algorithm for ℓ-pseudoforest Deletion

In this section we will present a $c_\ell^k n^{\mathcal{O}(1)}$ algorithm for ℓ-PSEUDOFOREST DELETION, where the constant c_ℓ depends only on ℓ. By Theorem 8.1, we know that ℓ-PSEUDOFOREST DELETION has a protrusion replacer. We first state the following theorem which will be used while designing the FPT algorithm as well as in the kernelization algorithm.

Theorem 8.3 (Linear Time Protrusion Replacement Theorem, [FLM+15]). *Let Π be a problem that has a protrusion replacer which replaces r protrusions of size at least r' for some fixed r. Let s and β be constants such that $s \geq r' \cdot 2^r$ and $r \geq 3(\beta+1)$. Given an instance (G, k) as input, there is an algorithm that runs in time $\mathcal{O}(m+n)$ and produces an equivalent instance (G', k') with $|V(G')| \leq |V(G)|$ and $k' \leq k$. If additionally G has a (α, β)-protrusion decomposition such that $\alpha \leq \frac{n}{244s}$, then we have that $|V(G')| \leq (1-\delta)|V(G)|$ for some constant $\delta > 0$.*

We call the algorithm in Theorem 8.3 Linear Time Protrusion Replacer (LPR). We show that given an instance (G, k) of ℓ-PSEUDOFOREST DELETION, there exists a constant ρ such that LPR can be used to get to an equivalent instance (G', k') where any ℓ-pseudoforest deletion set of G' is also an ρ-cover of G'.

Lemma 8.13. *There exist constants ρ, r, s and $c < 1$ such that if we run LPR with parameters r, s on an instance (G, k) of ℓ-*PSEUDOFOREST DELETION *such that G has an ℓ-pseudoforest deletion set S which is not a ρ-cover, then the output instance (G', k') satisfies $|V(G)| - |V(G')| \geq c|V(G)|$.*

Proof. Let S' be the ℓ-pseudoforest deletion set of G which is not a ρ cover. Let $S \subseteq S'$ be a minimal subset of S' which is an ℓ-pseudoforest deletion set.

By lemma 8.4, we know that $\mathbf{tw}(G - S) \leq (\ell + 1)$, and hence by lemma 8.12, G has a $(4(\ell + 2)|N[S]|, 2(\ell + 2))$-protrusion decomposition. We set $\beta = 2(\ell + 2)$, $r = 3(\beta + 1)$ and r' be the smallest such number for which the protrusion replacer for ℓ-PSEUDOFOREST DELETION replaces r protrusions of size at least r'. We fix $s = r' \cdot 2^r$. Now, we have a $(4(\ell+2)|N[S]|, \beta)$-protrusion decomposition of G. By Theorem 8.3, if $4(\ell+2)|N[S]| \leq \frac{n}{244s}$, then there exists a constant δ such that $|V(G)| - |V(G')| \geq \delta|V(G)|$. We set $c = \delta$ and show that there exists a constant $\rho < 1/3$ such that if S is not a ρ-cover of G, then $|N[S]| \leq \frac{n}{1000(\ell+2)s}$.

Since $\mathbf{tw}(G - S) \leq (\ell + 1)$, we have that $G - S$ is $(\ell + 2)$-degenerate. So, we have that $m \leq n(\ell + 2) + \sum_{v\in S} d(v)$. But we know that S is not a ρ cover, and also that it does not contain any isolated vertices since it is minimal. So we have that $|N[S]| \leq 2\sum_{v\in S} d(v) \leq 4\rho m$. Putting the inequalities together and rearranging, we have that $m \leq \frac{n(\ell+2)}{1-2\rho}$. This gives $|N[S]| \leq 4\rho m \leq 4\rho\frac{n(\ell+2)}{1-2\rho}$. Since we will fix $\rho < 1/3$, we have $|N[S]| \leq 12n\rho(\ell + 2)$. Choosing $\rho = \frac{1}{12000s(\ell+2)^2}$, we get $|N[S]| \leq \frac{n}{1000(\ell+2)s}$. Hence, there exist constants ρ and c depending only on ℓ such that if S is not a ρ-cover of G then the output instance (G', k) of LPR satisfies $|V(G)| - |V(G')| \geq c|V(G)|$. □

Lemma 8.14. *There is an algorithm that given an instance (G, k) of ℓ-*PSEUDOFOREST DELETION*, takes $\mathcal{O}((n + m)\log n)$ time and outputs an equivalent instance (G', k') such that $|V(G')| \leq |V(G)|$ and $k' \leq k$. Furthermore there exists a constant $0 < \rho < 1$ such that every ℓ-pseudoforest deletion set S of G' is a ρ-cover of G'.*

Proof. By Lemma 8.13 there exist constants ρ, r, s and $c < 1$ such that if we run the LPR with parameters r, s on an instance (G, k) such that G has an ℓ-pseudoforest deletion set S

which is not a ρ-cover, then the output instance (G', k') satisfies $|V(G)|-|V(G')| \geq c|V(G)|$. We set these constants as guaranteed by Lemma 8.13.

The algorithm sets $(G_0, k_0) := (G, k)$, $i := 0$ and enters a loop that proceeds as follows. The algorithm runs the LPR on (G_i, k_i) with parameters r and s, let the output of the LPR be (G_{i+1}, k_{i+1}). If $|V(G_i)| - |V(G_{i+1})| < c|V(G_i)|$ the algorithm halts and outputs (G_i, k_i). Otherwise, the algorithm increments i and returns to the beginning of the loop. One iteration of the loop takes time $O(|V(G_i)| + |E(G_i)|)$. Furthermore, every iteration of the loop reduces the number of vertices by a linear fraction. Hence, the total running time is bounded by $\mathcal{O}((m+n)\log n)$. Let (G', k') be the instance the algorithm outputs. By Lemma 8.13 we have that every ℓ-pseudoforest deletion set S of G' is a ρ-cover of G'. $\square$

Now we define the notion of *Buckets* which will be used by the algorithm crucially. Given graph G, we make a partition $P = \{B_1, B_2, \dots, B_{\lceil \log n \rceil}\}$ of $V(G)$ as follows.

$$B_i = \left\{ v \in V(G) \mid \frac{n}{2^i} < d(v) \leq \frac{n}{2^{i-1}} \right\} \text{ for all } i \in [\lceil \log n \rceil]$$

We call the sets B_i *buckets*. We call an instance (G, k) of ℓ-PSEUDOFOREST DELETION *irreducible* if applying the algorithm of Lemma 8.14 returns (G, k) itself, in which case, every ℓ-pseudoforest deletion set S of G is a ρ-cover of G. Let (G, k) be an irreducible instance of ℓ-PSEUDOFOREST DELETION and let X be an ℓ-pseudoforest deletion set of G of size at most k. A bucket B_i is said to be *good* if $|B_i \cap X| \geq d|B_i|$ and *big* if $|B_i| > i\lambda$, where d and λ are constants such that $(2d + 2\lambda) < \rho$.

Lemma 8.15. *Let (G, k) be an irreducible* YES *instance of ℓ-*PSEUDOFOREST DELETION. *Then G has a bucket that is both big and good.*

Proof. Since (G, k) is irreducible, every ℓ-pseudoforest deletion set X is a ρ-cover of G.

$$\sum_{v \in X} d(v) \geq 2m\rho$$

For contradiction, let us assume that G does not have a bucket which is both big and good.

```
Input:   A graph $G = (V, E)$ and positive integer $k$
Output:  YES, if $G$ has an $\ell$-pseudoforest deletion set of size at most $k$, No
         otherwise.
1 If $G$ is an $\ell$-pseudoforest then return YES. Else if $k \leq 0$ then return NO.
2 Apply Lemma 8.14 on $(G, k)$ to obtain an irreducible instance $(G', k')$.
3 Let $B_b = \{B_i \mid B_i \text{is big}\}$ be set of big buckets for $G'$. For every $B \in B_b$ and for every
  subset $S \subseteq B$ of size at least $d|B|$, check whether
  Algorithm-$\ell$-Pseudoforest-Modulator$(G' - S, k' - |S|)$ returns YES. If any of
  the calls returns YES, then return YES, otherwise return NO.
```
Algorithm 4: Algorithm-ℓ-Pseudoforest-Modulator(G, k)

Then we have the following.

$$\begin{aligned}
\sum_{v \in X} d(v) &= \sum_{i=1}^{\lceil \log n \rceil} \sum_{v \in B_i \cap X} d(v) \\
&\leq \sum_{\{i | B_i \text{ is not good}\}} \sum_{v \in B_i \cap X} d(v) + \sum_{\{i | B_i \text{ is not big}\}} \sum_{v \in B_i \cap X} d(v) \\
&\leq d \cdot 4m + \sum_{\{i | B_i \text{ is not big}\}} i\lambda \frac{n}{2^{i-1}} \\
&\leq d \cdot 4m + 4\lambda n \\
&\leq 2m(2d + 2\lambda)
\end{aligned}$$

We have chosen the values of d and λ such that $(2d+2\lambda) < \rho$, so we get $\sum_{v \in X} d(v) < 2m\rho$ which is a contradiction. □

Theorem 8.4. ℓ-PSEUDOFOREST DELETION *can be solved in time* $\mathcal{O}(c_\ell^k (m+n) \log n)$ *for an instance* (G, k) *where the constant* c_ℓ *depends only on* ℓ.

Proof. We use Algorithm 4 for solving the problem. The algorithm first applies Lemma 8.14 to get to an irreducible graph G' and then branches on all possible big subsets of the big buckets. The correctness follows from Lemma 8.15. Now we analyze the running time of the algorithm. We first look at the case where all buckets are big. Let a_i be the size of the bucket i. Then we have the following recurrence relation.

$$T(k) \leq \sum_{i=1}^{\lceil \log n \rceil} 2^{a_i} T(k - da_i)$$

Assuming $T(k) = x^k$ and by substituting we get the following.

$$\begin{aligned} T(k) &\leq \sum_{i=1}^{\lceil \log n \rceil} 2^{a_i} x^{k-da_i} \\ &\leq x^k \sum_{i=1}^{\lceil \log n \rceil} \left(\frac{2}{x^d}\right)^{a_i} \end{aligned}$$

Now, if $\frac{2}{x^d} < 1$, then the summation is maximized when the exponent of $\frac{2}{x^d}$ is minimized. Since the buckets chosen are big, we have that $a_i \geq \lambda i$. Hence, for values of x such that $\frac{2}{x^d} < 1$, we get the following.

$$T(k) \leq x^k \sum_{i=1}^{\lceil \log n \rceil} \left(\frac{2}{x^d}\right)^{\lambda i}$$

The sum $\sum_{i=1}^{\lceil \log n \rceil} \left(\frac{2}{x^d}\right)^{\lambda i}$ is a geometric series, which converges to 1 for some $x = c_\ell$, where the constant c_ℓ is suitably large and depends only upon d and λ. But we know that d and λ depend only on ρ which in turn depends only on ℓ. So this bounds the running time of the algorithm by c_ℓ^k, where the constant c_ℓ depends only on ℓ.

In the case where not all buckets are big, algorithm goes through only the big buckets and the summation is done only for the big buckets. Hence, even in this case, the running time is bounded by the same function. $\square$

8.4 A $c^k n^{\mathcal{O}(1)}$ algorithm for Pseudoforest Deletion

In this section we will present a $c^k n^{\mathcal{O}(1)}$ algorithm for Pseudoforest Deletion. We use the well known technique of iterative compression, and arrive at the desired running time after defining a non-trivial measure.

Given an instance (G, k) of Pseudoforest Deletion, let $V(G) = \{v_1, \dots, v_n\}$ and define vertex sets $V_i = \{v_1, \dots, v_i\}$, and let the graph $G_i = G[V_i]$. We iterate through the instances (G_i, k) starting from $i = k + 1$. For the i^{th} instance, we try to find a pseudoforest deletion set $\hat{S}_i$ of size at most k, with the help of a *known* pseudoforest deletion set S_i of size at most $k + 1$. Formally, the compression problem we address is the following.

PSEUDOFOREST DELETION COMPRESSION **Parameter(s):** k

Input: A graph G, and pseudoforest deletion set of G of size at most $k+1$, integer k.

Question: Does there exist a pseudoforest deletion set for G of size k?

Lemma 8.16. *If* PSEUDOFOREST DELETION COMPRESSION *can be solved in $f(k)n^c$ time, then* PSEUDOFOREST DELETION *can be solved in $f(k)n^{c+1}$ time.*

Proof. We solve the PSEUDOFOREST DELETION problem by iteratively solving at most $n-k$ instances of the PSEUDOFOREST DELETION COMPRESSION problem as follows. Let $I_i = (G_i, S_i, k)$ be the i^{th} instance. Clearly, the set V_{k+1} is a pseudoforest deletion set of size at most $k+1$ for the instance I_{k+1}. It is also easy to see that if $\hat{S}_{i-1}$ is a pseudoforest deletion set of size at most k for instance I_{i-1}, then the set $\hat{S}_{i-1} \cup \{v_i\}$ is a pseudoforest deletion set of size at most $k+1$ for the instance I_i. We use these two observations to start off the iteration with the instance $(G_{k+1}, S_{k+1} = V_{k+1}, k)$ and look for a pseudoforest deletion set of size at most k for this instance. If there is such a pseudoforest deletion set $\hat{S}_{k+1}$, we set $S_{k+2} = \hat{S}_{k+1} \cup \{v_{k+2}\}$ and ask for a pseudoforest deletion set of size at most k for the instance I_{k+2} and so on. If, during any iteration, the corresponding instance does not have a pseudoforest deletion set of the required size, it implies that the original instance is also a NO instance. This follows from the fact that if a graph G has a pseudoforest deletion set of size at most k, then any vertex induced subgraph of G also has a pseudoforest deletion set of size at most k. Finally, the solution for the original input instance will be $\hat{S}_n$. Since there can be at most n iterations, the total time taken to solve the original instance is bounded by n times the time required to solve the PSEUDOFOREST DELETION COMPRESSION problem. □

For designing an algorithm for PSEUDOFOREST DELETION COMPRESSION, let the input instance be (G, S, k). We guess a subset $Y \subseteq S$ with the intention of picking these vertices in our hypothetical solution for this instance and not picking the rest of the vertices in S in the solution. We delete the set Y from the graph and decrease k by $|Y|$. We then check if the graph $G[S \setminus Y]$ is a pseudoforest and if it is not, then reject this guess of Y as a spurious guess. Suppose that $G[S \setminus Y]$ is indeed a pseudoforest. Hence, it now remains to check if there is a pseudoforest deletion set S' of size $k' = k - |Y|$ which is disjoint from $S \setminus Y$, and $G - (Y \cup S')$ is a pseudoforest. More precisely, we have an instance of Pseudoforest Deletion Disjoint Compression, which is defined as follows.

Pseudoforest Deletion Disjoint Compression	**Parameter(s):** k
Input: A graph G, a pseudoforest deletion set S of G, integer k	
Question: Does there exist a pseudoforest deletion set of G disjoint from S of size at most k?	

To solve the problem, we first design a set of reduction rules, and prove some lemmata about their correctness.

Reduction Rule 13. *If there exists a vertex v of degree at most 1 in the graph, delete it.*

Reduction Rule 14. *If there exists $v \in V(G) \setminus S$ such that $G[S \cup \{v\}]$ is not a pseudoforest, delete v and decrease k by 1.*

Reduction Rule 15. *If there exists a vertex $v \in V(G) \setminus S$ of degree two in G, such that at least one of its neighbours are in $V(G) \setminus S$, delete v, and put an edge between its neighbours (even if they were already adjacent). If both of its edges are to the same vertex, delete v and put a self loop on the adjacent vertex (even if it has self loop(s) already).*

Now we proceed towards proving the correctness of these reduction rules. Let (G, S, k) be the instance on which the reduction rule is being applied and let (G', S', k') be the instance after the application of the reduction rule. We say that a reduction rule is *correct* if (G, S, k) is a Yes instance of Pseudoforest Deletion Disjoint Compression if and only if (G', S', k') is a Yes instance. The correctness of Reduction Rule 13 is obvious from Lemma 8.3.

Lemma 8.17. *Reduction Rule 14 is correct.*

Proof. For showing correctness of this reduction rule, we just need to show that v is part of every pseudoforest deletion set of G of size at most k, which is disjoint from S. This is indeed true, because if not so, then we know that $G[S \cup \{v\}]$ is not a pseudoforest. Hence for any set X of size at most k disjoint from S which does not contain v, $G - X$, being a supergraph of $G[S \cup \{v\}]$, is not a pseudoforest as well. □

Lemma 8.18. *Reduction Rule 15 is correct.*

Proof. Let (G, S, k) be a Yes instance of Pseudoforest Deletion Disjoint Compression, and let X be a pseudoforest deletion set of G of size at most k, which is disjoint from S. Let v be the vertex of degree 2 being deleted from the graph. Let us first examine the case where v has two distinct neighbours x and y. If $X \cap \{v, x, y\} \neq \emptyset$, then we put $X' = (X \setminus \{v\}) \cup \{x\}$ if $v \in X$ or $X' = X$ if $v \notin X$. It is easy to see that $|X'| \leq k$ and $G' - X'$ is a subgraph of $G - X$ in both the cases and hence $G' - X'$ is a pseudoforest. On the other hand, if none of v, x or y are in X, then we look at the connected component of $G' - X$ containing x and y. This is the only connected component for which the excess could have possibly changed. But this connected component loses one vertex and two edges, while gaining one new edge. So, the excess for this component also remains the same, and hence $G' - X$ is a pseudoforest. The case when both of v's edges are to the same vertex is similar.

For the converse, let (G', S, k) be a Yes instance of Pseudoforest Deletion Disjoint Compression which we get after applying Reduction Rule 15 and let X be a pseudoforest deletion set of G' of size at most k, which is disjoint from S. If X contains any neighbour of v, then in $G - X$, v has degree at most 1 and hence the edge incident on it (if it survives) does not contribute to excess of any connected component. So, $G - X$ remains to be a pseudoforest. In the case where none of v's neighbours belong to X, only the connected component including the neighbours can possibly gain from introducing v back into the graph. But in this case, since we delete the edge between the neighbours (or the self loop) to get to G, the component gains exactly one vertex and one edge, and hence even in this case $G - X$ is a pseudoforest. So, (G, S, k) is a Yes instance of Pseudoforest Deletion Disjoint Compression. This concludes the proof of the lemma. $\square$

We say that an instance (G, S, k) of Pseudoforest Deletion Disjoint Compression is a *good* instance if $G - S$ is a disjoint union of cycles such that for all $v \in V(G) \setminus S$, v has exactly one neighbour in S and $G[S]$ has no connected components which are trees i.e. all the connected components of $G[S]$ have a cycle. We first show that Pseudoforest Deletion Disjoint Compression is polynomial time solvable on good instances.

Lemma 8.19. Pseudoforest Deletion Disjoint Compression *can be solved in polynomial time on good instances.*

Proof. Let (G, S, k) be a good instance of Pseudoforest Deletion Disjoint Compression. Then we want to show that (G, S, k) is a Yes instance if and only if $G - S$ has a vertex cover of size at most k.

For the forward direction, let X be a pseudoforest deletion set of G disjoint from S of size at most k. We claim that X is a vertex cover for $G - S$. For the sake of contradiction we assume it is not, then there exist $u, v \in V(G) \setminus S$ such that $(u, v) \in E(G)$ and $u, v \notin X$. But we also know that u and v both have a neighbour in S. Let x and y be the neighbours of u and v and let S_1 and S_2 be the connected components of $G[S]$ containing x and y respectively (it is possible that $x = y$ or $S_1 = S_2$). Let $S' = S_1 \cup S_2 \cup \{u, v\}$. Since $S \cap X = \emptyset$, $G[S']$ is a connected subgraph of $G - X$ and $E(G[S']) = |S' + 1| > |S'|$. Hence, $G[S']$ is not a pseudoforest, and neither is $G - X$, which contradicts our assumption.

For the converse, let X be a vertex cover for $G - S$. Let $S' = (V(G) \setminus (S \cup X))$. Then we have that $G[S']$ is an independent set. Since every vertex $v \in S'$ has exactly one neighbour in S, this means that all the vertices $v \in S'$ have degree exactly one in $G[S \cup S']$. We know that $G[S]$ is a pseudoforest (otherwise we would have rejected the instance as a spurious one) and hence $G[S \cup S']$ is also a pseudoforest by Lemma 8.3 and X is a pseudoforest deletion set of G disjoint from S of size at most k.

Finally, the minimum vertex cover of $G - S$ can be found in polynomial time since $G - S$ is a disjoint union of cycles. ☐

Lemma 8.20. *After the exhaustive application of Reduction Rules 13, 14 and 15, either there exists $v \in V(G) \setminus S$ such that v has at least two neighbours in S or $G - S$ is a disjoint union of cycles.*

Proof. We know that $G - S$ is a pseudoforest. If $G - S$ is not a disjoint union of cycles, then there exists $v \in V(G) \setminus S$ such that $d_{G-S}(v) \leq 1$. If $d_{G-S}(v) = 0$, then we know that v has at least two neighbours in S, as otherwise Reduction Rule 13 would have applied.

```
Input:   A graph $G = (V, E)$, $S \subseteq V$ and positive integer $k$
Output:  YES, if $H$ has a pseudoforest deletion set of size at most $k$ disjoint from $S$,
         No otherwise.
1 If $G[S]$ or $G - S$ is not a pseudoforest, return NO.
2 If $G$ is a pseudoforest then return YES. Else if $k \leq 0$ then return NO.
3 Apply Reduction Rule 13 or 14 or 15, whichever applicable, to get an instance
  $(G', S', k')$. Return PFD$(G', S', k')$.
4 If $(G, k)$ is a good instance, then solve the problem in polynomial time.
5 If $G - S$ has a vertex $v$ which has at least two neighbours in $S$, return
  PFD$(G - \{v\}, S, k - 1)\lor$ PFD$(G, S \cup \{v\}, k)$.
6 Else pick any vertex $v \in V(G) \setminus S$
7 Case 1. $v$ has a self loop, return PFD$(G - \{v\}, S, k - 1)\lor$ PFD$(G, S \cup \{v\}, k)$.
8 Case 2. $v$ is part of a cycle of length 2, return PFD$(G - \{v\}, S, k - 1)\lor$
  PFD$(G - \{u\}, S \cup \{v\}, k - 1)\lor$ PFD$(G, S \cup \{u, v\}, k)$, where $u$ is the other vertex in
  the cycle.
9 Case 3. $v$ is part of a cycle of length $\geq 3$, return PFD$(G - \{v\}, S, k - 1)\lor$
  PFD$(G - \{u, w\}, S \cup \{v\}, k - 2)\lor$ PFD$(G - \{w\}, S \cup \{u, v\}, k - 1)\lor$
  PFD$(G - \{u\}, S \cup \{v, w\}, k - 1)\lor$ PFD$(G, S \cup \{u, v, w\}, k)$, where $u$ and $w$ are $v$'s
  neighbours in the cycle.
```

Algorithm 5: PFD(G, S, k)

If $d_{G-S}(v) = 1$, then again we have that v has at least two neighbours in S, as otherwise Reduction Rule 13 or 15 would have applied. □

Now we are ready to describe our algorithm for PSEUDOFOREST DELETION DISJOINT COMPRESSION. A pseudocode of the same can be found as Algorithm 5. If $G[S]$ or $G - S$ is not a pseudoforest, then we reject the instance as a spurious one. If $k < 0$, then we return that the given instance is a NO instance. Otherwise, we first apply the reduction rules 13, 14 and 15 exhaustively. If the input instance (G, S, k) is a good instance of PSEUDOFOREST DELETION DISJOINT COMPRESSION, then we solve it by computing a vertex cover of $G - S$, which can be done in polynomial time. Now we look at two possible cases.

1. There exists a vertex $v \in V(G) \setminus S$ which has at least two neighbours in S. We branch by either including v in our solution or excluding it. More precisely, we call the algorithm recursively on $(G - \{v\}, S, k - 1)$ and $(G, S \cup \{v\}, k)$. If at least one of the recursive call returns YES, then we say that the instance was a YES instance.

2. There does not exist a vertex $v \in V(G) \setminus S$ which has at least two neighbours in S. In this case, we know by Lemma 8.20 that $G - S$ is a disjoint union of cycles. We also have that for all $v \in V(G) \setminus S$, v has exactly one neighbour in S, because otherwise Reduction Rule 15 would have applied. Let $\mathcal{T}$ be the set of connected components of $G[S]$ which are trees, and let $\mathcal{C}$ be the set of remaining connected components (which have a cycle). Since the instance is not a good instance, $G[S]$ has at least one connected component $T \in \mathcal{T}$. Since Reduction Rule 13 does not apply, there must exist $v \in V(G) \setminus S$ such that v has a neighbour in T. We look at the three possible cases depending upon length of the cycle v is part of, and describe the branching for each of them.

1. **v has a self loop.** In this case, we branch on v by including it in our solution or excluding it. That is, we call the algorithm recursively on $(G - \{v\}, S, k - 1)$ and $(G, S \cup \{v\}, k)$.

2. **v is part of cycle of length 2.** In this case, we do a three-way branching. The first branch deals with the case when we v is part of the solution. Let u be the other vertex in the cycle v is part of. In the other case, where v is not part of the solution, we divide it into two more branches depending upon whether u is part of the solution. We denote these branches by (v) $(\bar{v}, u)$, $(\bar{v}, \bar{u})$, were (v) denotes the branch where v is part of the solution, $(\bar{v}, u)$ denotes the branch where v is not part of the solution but u is, and $(\bar{v}, \bar{u})$ denotes the solution where v and u both are not part of the solution. The detailed description of the branches is as following.

    (a) $(\mathbf{v})$: We call the algorithm recursively on $(G - \{v\}, S, k - 1)$.

    (b) $(\bar{\mathbf{v}}, \mathbf{u})$: We call the algorithm recursively on $(G - \{u\}, S \cup \{v\}, k - 1)$

    (c) $(\bar{\mathbf{v}}, \bar{\mathbf{u}})$: We call the algorithm recursively on $(G, S \cup \{u, v\}, k)$

3. **v is part of a cycle of length at least 3.** In this case we do a five-way branching. Let u and w be the vertices which are adjacent to v in the cycle which contains v in $G - S$. The first branch corresponds to the case when v is part of the solution. In the other case, where v is not part of the solution, we divide it into four more branches depending upon whether u are w are part of the solution. We denote these branches by (v), $(\bar{v}, u, w)$, $(\bar{v}, \bar{u}, w)$, $(\bar{v}, u, \bar{w})$ and $(\bar{v}, \bar{u}, \bar{w})$, using the same notation as in the previous case. The detailed description of the branches is as following.

(a) $(\mathbf{v})$: We call the algorithm recursively on $(G - \{v\}, S, k - 1)$.

(b) $(\bar{\mathbf{v}}, \mathbf{u}, \mathbf{w})$: We call the algorithm recursively on $(G - \{u, w\}, S \cup \{v\}, k - 2)$.

(c) $(\bar{\mathbf{v}}, \bar{\mathbf{u}}, \mathbf{w})$: We call the algorithm recursively on $(G - \{w\}, S \cup \{u, v\}, k - 1)$

(d) $(\bar{\mathbf{v}}, \mathbf{u}, \bar{\mathbf{w}})$: We call the algorithm recursively on $(G - \{u\}, S \cup \{v, w\}, k - 1)$

(e) $(\bar{\mathbf{v}}, \bar{\mathbf{u}}, \bar{\mathbf{w}})$: We call the algorithm recursively on $(G, S \cup \{u, v, w\}, k)$

This concludes the description of the algorithm. The correctness of the algorithm follows from the correctness of reduction rules and the fact that the branching is exhaustive.

To analyze the running time of the algorithm, we define a measure $\phi(I)$ for the input instance $I = (G, S, k)$ as follows.

$$\phi(I) = k + \mathsf{tc}(S)$$

Where $\mathsf{tc}(S)$ denotes the number of connected components of $G[S]$ which are trees.

Lemma 8.21. *None of the reduction rules increase the measure $\phi(I)$.*

Proof. Reduction Rule 13 deletes a vertex of degree at most 1 from the graph. It does not affect k and it might decrease the number of connected components in $G[S]$, but the measure $\phi(I)$ does not increase. Reduction Rule 14 deletes a vertex v for which $G[S \cup \{v\}]$ is not a pseudoforest, and decreases k by 1. It does not modify the set S and hence does not change $\mathsf{tc}(S)$. So, the measure drops by at least 1. For Reduction Rule 15, k and $\mathsf{tc}(S)$ remain the same. So, we have that none of the reduction rules increases the measure. □

Now, we look at the branching steps to analyze the running time of the algorithm. We look at all possible cases.

1. Branching on a vertex v which has at least two neighbours in S. In one branch, we call the algorithm on $(G - \{v\}, S, k - 1)$. It is easy to see that in this branch $\mathsf{tc}(S)$ remains the same while k decreases by 1, and hence the measure drops by 1. In the other branch, we call the algorithm recursively on $(G, S \cup \{v\}, k)$ and hence k does not change. Let $S' = S \cup \{v\}$. Since v has at least two neighbour in S, let us call any two of them x and y which belong to connected components C_1 and C_2 respectively. If $C_1, C_2 \in \mathcal{C}$, then Reduction Rule 14 would have applied. Hence, at least one of C_1 and C_2 is a tree. After adding v to the set S, $C_1 \cup C_2 \cup \{v\}$ is part of the same connected component. Hence, it is easy to see that $\mathsf{tc}(S \cup \{v\}) \le \mathsf{tc}(S) - 1$ and the measure drops by at least 1. So, in the worst case, we get a branching factor of $(1, 1)$.

2. Branching on a vertex v which is part of a cycle and has exactly one neighbour in S. Let $\mathcal{T}$ be the set of connected components of $G[S]$ which are trees, and let $\mathcal{C}$ be the set of remaining connected components (which have a cycle). Let T be a connected component of $G[S]$ such that v has a neighbour in T, then from the description of the algorithm, we know that $T \in \mathcal{T}$. As we did earlier, we divide this case into three sub-cases on the basis of the size of the cycle v is part of.

1. v **has a self loop.** In the first branch, we call the algorithm on $(G - \{v\}, S, k - 1)$. In this branch k drops by 1 while $\mathsf{tc}(S)$ remains the same and hence the measure drops by 1. Let C be connected component of S in which v has a neighbour and let $S' = S \cup \{v\}$. Clearly, $C \notin \mathcal{C}$ because then Reduction Rule 14 would apply. Hence, we have that $C \in \mathcal{T}$. All the connected components of S' are the same except the one which was a tree in S but now contains v and hence the loop. Hence, $\mathsf{tc}(S') = \mathsf{tc}(S) - 1$ and the measure drops by 1. We get a branching factor of $(1, 1)$.

2. v **is part of a cycle of length** 2. This case results in a three way branching. We analyze the measure drop in all of them one by one.

   (a) $(\mathbf{v})$: We call the algorithm recursively on $(G - \{v\}, S, k - 1)$. So k drops by 1 and $\mathsf{tc}(S)$ remains the same and hence the measure decreases by 1.

   (b) $(\bar{\mathbf{v}}, \mathbf{u})$: We call the algorithm recursively on $(G - \{u\}, S \cup \{v\}, k - 1)$, hence k decreases by 1. Let $S' = S \cup \{v\}$. Since v has at exactly one neighbour in S, $\mathsf{tc}(S \cup \{v\}) = \mathsf{tc}(S)$ and hence the measure drops by 1.

   (c) $(\bar{\mathbf{v}}, \bar{\mathbf{u}})$: We call the algorithm recursively on $(G, S \cup \{u, v\}, k)$. Observe that if u has a neighbour in a connected component C of $G[S]$ where $C = T$ or $C \in \mathcal{C}$, then $G[C \cup \{u, v\}]$ has $|C| + 3$ edges and recursive call returns No in polynomial time. Otherwise, u has neighbour in some $T' \in \mathcal{T}$ such that $T' \ne T$.

Let $S' = S \cup \{u, v\}$, then $G[T \cup T' \cup \{u, v\}]$ is a connected component with a cycle (the parallel edge between u and v) in $G[S']$. All other tree components of $G[S]$ remain the same and we have $\mathsf{tc}(S') = \mathsf{tc}(S) - 2$. Hence, the measure drops by 2 in this branch.

Hence, we get a branching factor of $(1, 1, 2)$ in the worst case when v is part of a cycle of length 2.

3. v **is part of a cycle of length at least** $\mathbf{3}$**.** This case results in a five way branching. We look at all of them one by one to analyze the measure drop.

   (a) $(\mathbf{v})$: In this case, the algorithm gets called on $(G - \{v\}, S, k - 1)$, so k drops by 1 while $\mathsf{tc}(S)$ remains the same, and hence the measure drops by 1.

   (b) $(\bar{\mathbf{v}}, \mathbf{u}, \mathbf{w})$**.** The algorithm is called recursively on $(G - \{u, w\}, S \cup \{v\}, k - 2)$, so k drops by 2. Let $S' = S \cup \{v\}$. Since v has at exactly one neighbour in S, $\mathsf{tc}(S \cup \{v\}) = \mathsf{tc}(S)$ and hence the measure drops by 2.

   (c) $(\bar{\mathbf{v}}, \bar{\mathbf{u}}, \mathbf{w})$**.** The algorithm is called recursively on $(G - \{w\}, S \cup \{u, v\}, k - 1)$ and hence k decreases by 1. Let $S' = S \cup \{v, u\}$. Since v has a neighbour in some $T \in \mathcal{T}$, due to the edge (v, u) getting pushed to S this tree component in $G[S]$ is either not a tree component in $G[S']$ or gets connected to a different connected component of $G[S]$ depending on whether u has a neighbour in T or not. In any case, we have that $\mathsf{tc}(S') = \mathsf{tc}(S) - 1$ and hence the measure drops by 2.

   (d) $(\bar{\mathbf{v}}, \mathbf{u}, \bar{\mathbf{w}})$**.** This case is symmetric to the previous one and hence we get a drop of 2 in the measure.

   (e) $(\bar{\mathbf{v}}, \bar{\mathbf{u}}, \bar{\mathbf{w}})$**.** In this case, the algorithm gets called recursively on $(G, S \cup \{u, v, w\}, k)$ and hence k remains the same. Let $S' = S \cup \{v, u, w\}$. We observe that if u and w have a neighbour in C_1 and C_2 respectively such that $C_1, C_2 \in \mathcal{C}$ (note that it is possible that $C_1 = C_2$), then the recursive call returns No in polynomial time. This is because then $G[S']$ will not be a pseudoforest. Otherwise, without loss of generality, let u have a neighbour in connected component $T' \in \mathcal{T}$. If $T = T'$, then if w has a neighbour in a connected component $C \in \mathcal{C} \cup \{T\}$, then the recursive call returns NO, because then $G[S']$ is not a pseudoforest. If w has a neighbour in some other tree component, then we see that $\mathsf{tc}(S') = \mathsf{tc}(S) - 2$, because both the tree components become part of one component which has a cycle. Hence we have a measure drop of 2. The remaining case is where $T \neq T'$. Now, if w has a neighbour in either T or T', then T and T' are part of a single component in $G[S']$ which is not a tree. Hence $\mathsf{tc}(S') = \mathsf{tc}(S) - 2$. On the other hand, if w has a neighbour in T_2 such that $T_2 \neq T$ and $T_2 \neq T_1$, then in $G[S']$, we have a single component which contains T, T_1 and T_2 and hence $\mathsf{tc}(S') = \mathsf{tc}(S) - 2$. So in any case we have a drop of 2 in $\mathsf{tc}(S')$ and in the measure as well.

So, we get a branching factor of $(1, 2, 2, 2, 2)$ in the case when v is part of a cycle of length at least 3.

Lemma 8.22. PSEUDOFOREST DELETION DISJOINT COMPRESSION *can be solved in* $\mathcal{O}^*(6.5618^k)$ *time.*

Proof. We know that initially $|S| \leq k+1$ and hence $0 \leq \mathsf{tc}(S) \leq k+1$. This proves that $0 \leq \phi(I) \leq 2k+2$ initially. Also, if $\phi(I) \leq 0$ during the course of the algorithm, then we conclude that $k \leq 0$ and solve the problem trivially. We have seen that none of the reduction rules increase the measure. It is easy to see that the reduction rules can be applied in polynomial time.

For the branching part, we have three branching factors, $(1, 1)$, $(1, 1, 2)$ and $(1, 2, 2, 2, 2)$. Also, we have shown that at each branching step, in polynomial time either solves the problem, or makes a recursive call. Hence, the time spent at each branching step is polynomial. Out of the three branching factors, $(1, 2, 2, 2, 2)$ is the worst and gives a running time of $(6.5618)^k n^{\mathcal{O}(1)}$. □

Given Lemma 8.22, the algorithm for PSEUDOFOREST DELETION COMPRESSION has a running time of $\mathcal{O}(\sum_{i=0}^{k} \binom{k+1}{i} \cdot (6.5618)^k \cdot n^{\mathcal{O}(1)}) = \mathcal{O}^*(7.5618^k)$. Here, the factor of $\binom{k+1}{i}$ is for the guesses we make for the set S. After this, applying Lemma 8.16 gives the following theorem.

Theorem 8.5. Pseudoforest Deletion *can be solved in* $\mathcal{O}^*(7.5618^k)$ *time.*

8.5 Kernels

In this section, we give kernelization algorithm for ℓ-pseudoforest Deletion. First, we give a procedure to reduce the maximum degree of the graph. Then we arrive at a kernel by exploiting the small maximum degree using protrusions. In the end, we give an explicit kernel for Pseudoforest Deletion.

8.5.1 Degree Reduction

For bounding the maximumum degree of the graph, we give a set of reduction rules which apply in polynomial time. If the maximum degree of the graph is already bounded by $(k+\ell)(3\ell+8)$, then we have nothing to do. Hence, for the rest of this subsection, we assume that there exists a vertex with degree greater than $(k+\ell)(3\ell+8)$.

Reduction Rule 16. *If there exists a vertex v of degree at most 1 in the graph, delete it.*

Correctness of Reduction Rule 16 follows from Lemma 8.3.

Reduction Rule 17. *If any edge has multiplicity more that $\ell+2$, then delete all but $\ell+2$ copies of that edge.*

Lemma 8.23. *Reduction Rule 17 is correct.*

Proof. Let (u, v) be the edge with multiplicity more than $\ell+2$ in G. Since G' is a subgraph for G, any solution X for G of size at most k will also be a solution for G'. So, if (G, k) is a Yes instance, then so is (G', k). For the converse, let (G', k) be a Yes instance. Any ℓ-pseudoforest deletion set X for G' must contain either u or v, otherwise we have that $G[\{u, v\}]$ is not an ℓ-pseudoforest, which is a subgraph of $G - X$, and hence $G - X$ is also not an ℓ-pseudoforest. Since we have only altered edges between u and v and X contains at least one of them, $G - X$ is identical to $G' - X$. Hence, $G - X$ is an ℓ-pseudoforest and (G, k) is a Yes instance. □

Lemma 8.24. *Any connected component in an ℓ-pseudoforest can have at most ℓ edge disjoint cycles.*

Proof. Let the connected component C of the graph G have ℓ' edge disjoint cycle such that $\ell' > \ell$. To get a forest as a subgraph of G, we must delete at least one edge from every cycle. Since the cycles are edge disjoint, we need to delete at least $\ell' > \ell$ edges from C to get to a forest. This shows that $\mathsf{ex}(C) > \ell$ and hence G is not an ℓ-pseudoforest. □

We need one more reduction rule before we proceed further.

Reduction Rule 18. *If there is a vertex v with more than ℓ self loops, delete v and decrease k by one.*

Correctness of the reduction rule follows immediately from lemma 8.24.

We now look at a vertex which, after application of reduction rules 16-18, still has high degree. The idea is that either a high degree vertex participates in many cycles (and contributes many excess edges) and hence should be part of the solution, or only a small part of its neighbourhood is relevant for the solution. We formalize these notions as similar to the case of FVS and Almost Forest Deletionand use them to find flowers using Gallai's theorem and apply a set of reduction rules. Given a set $T \subseteq V(G)$, by T-path we mean set of paths of positive length with both endpoints in T. Let us recall the statement of Gallai's theorem once more.

Theorem 8.6 (Gallai, [Gal64]). *Given a simple graph G, a set $T \subseteq V(G)$ and an integer s, one can in polynomial time find either*

- *a family of $s+1$ pairwise vertex-disjoint T-paths, or*
- *a set B of at most $2s$ vertices, such that in $G - B$ no connected component contains more than one vertex of T.*

As in the previous chapter, we would want to have the neighborhood of a high degree vertex as the set T for applying Gallai's theorem and for detecting flowers. We first take care of multiple edges and self loops. Let v be a vertex with high degree. The vertices in

$N(v)$ which have at least two parallel edges to v can be greedily picked to form a petal of the flower. Let L be the set of vertices in $N(v)$ which have at least two parallel edges to v.

Reduction Rule 19. *If $|L| > k + \ell$, delete v and decrease k by 1.*

Lemma 8.25. *Reduction Rule 19 is correct.*

Proof. The correctness of the reduction rule follows from the fact that any ℓ-pseudoforest deletion set of size at most k must delete v. This is true because otherwise, we have at least $\ell + k + 1$ petals which form edge disjoint cycles, and hence after deleting any of the k vertices from $N(v)$, we will be left with at least $\ell + 1$ edge disjoint cycles in a connected component of G. So, by Lemma 8.24, the graph will not be an ℓ-pseudoforest. $\square$

Let $\widehat{G}$ be the graph $G - L$ with all parallel edges replaced with single edges, and all self loops removed. Now we apply Gallai's theorem on $\widehat{G}$ with $T = N(v)$ and $s = k + \ell - |L|$. If the theorem returns a collection of vertex disjoint T-paths, then it is easy to see that they are in one to one correspondence with cycles including v, and hence can be considered petals of the flower centered at v.

Reduction Rule 20. *If the application of Gallai's theorem returns a flower with more than s petals, then delete v and decrease k by 1.*

Lemma 8.26. *Reduction Rule 20 is correct.*

Proof. Let us assume that the application of Gallai's theorem on $\widehat{G}$ returns a flower centered at v with at least $k + \ell + 1 - |L|$ petals. Since $\widehat{G}$ does not contain L, we can add a pair of parallel edges of the form (x, v) for all $x \in L$ to form a flower centered at v with at least $k + \ell + 1$ petals in G. This forces v to be part of any ℓ-pseudoforest deletion set of G of size at most k. $\square$

Now, we deal with the case when the application of Gallai's theorem returns a set B of at most $2(k + \ell - |L|)$ vertices, such that in $G - B$, no connected component contains more than one vertex of T. Let $Z = B \cup L$. Clearly, $|Z| \leq 2(k + \ell) - |L|$. Now we look at the set of connected components of $G - (Z \cup \{v\})$. Let us call this set C. We first show that if too many connected components in C have a cycle, then v has to be part of any ℓ-pseudoforest deletion set of G of size at most k.

Reduction Rule 21. *If more than $k + \ell$ components of C contain a cycle, then delete v and reduce k by one.*

Lemma 8.27. *Reduction rule 21 is correct.*

Proof. For proving the correctness of the reduction rule, we just need to show that if more than $k + \ell$ components of C contain a cycle then v is part of any ℓ-pseudoforest deletion set of G of size at most k. Let X be an ℓ-pseudoforest deletion set of G of size at most k and $v \notin X$. Now, we know that all the cycles in different connected components of C are vertex disjoint. Deleting any set X of size at most k leave at least $\ell + 1$ cycles in the graph which are vertex disjoint and hence edge disjoint. Also, all these $\ell + 1$ connected components of C containing a cycle are connected to v and we get a connected component in $G - X$ with more than ℓ edge disjoint cycles. By Lemma 8.24, we know that $G - X$ is not an ℓ-pseudoforest, which is a contradiction. ☐

Lemma 8.28. *After applying reduction rules 16-21 exhaustively, there are at least $2(\ell + 2)(k + \ell)$ components in C which are trees and are connected to v with exactly one edge.*

Proof. The number of self loops on v is bounded by ℓ due to Reduction Rule 18. Number of edges from v to Z is bounded by $|B| + (\ell + 2)|L| \leq 2(k + \ell - |L|) + (\ell + 2)|L| = 2(k + \ell) + \ell|L| \leq (k + \ell)(\ell + 2)$. As degree of v is greater than $(k + \ell)(3\ell + 8)$, at least $(k+\ell)(3\ell+8) - (k+\ell)(\ell+2) - \ell \geq (k+\ell)(2\ell+5)$ connected components in C have exactly one vertex which is is neighbour of v. Out of these, the number of connected components containing cycles is bounded by $k + \ell$ by Reduction Rule 21. Hence, at least $2(\ell + 2)(k + \ell)$ connected components are trees and are connected to v by exactly one edge. ☐

Before we proceed further, we state the Expansion Lemma. Let G be a bipartite graph with vertex bipartition (A, B). For a positive integer q, a set of edges $M \subseteq E(G)$ is called a q-expansion of A into B if every vertex of A is incident with exactly q edges of M, and exactly $q|A|$ vertices in B are incident to M.

Lemma 8.29 (Expansion Lemma, [FLM+11])**.** *Let $q \geq 1$ be a positive integer and G be a bipartite graph with vertex bipartition (A, B) such that $|B| \geq q|A|$ and there are no isolated vertices in B. Then there exist nonempty vertex sets $X \subseteq A$ and $Y \subseteq B$ such that there is a q-expansion of X into Y and no vertex in Y has a neighbor outside X, that is, $N(Y) \subseteq X$. Furthermore, the sets X and Y can be found in time polynomial in the size of G.*

Let D the set of connected components which are trees and connected to v with exactly one edge. We have shown that $|D| \geq 2(\ell + 2)(k + \ell)$. Now we construct an auxiliary bipartite graph H as follows. In one partition of H, we have a vertex for every connected component in D, and the other partition is Z. We put an edge between $A \in D$ and $v \in Z$ if some vertex of A is adjacent to v. Since every connected component in D is a tree and has only one edge to v, some vertex in it has to have a neighbour in Z, otherwise Reduction Rule 16 would apply. Now we have that $|Z| \leq 2(k + \ell)$ and every vertex in D is adjacent to some vertex in Z, we may apply q-expansion lemma with $q = \ell + 2$. This means, that in polynomial time, we can compute a nonempty set $\widehat{Z} \subseteq Z$ and a set of connected components $\widehat{D} \subseteq D$ such that:

1. $N_G(\bigcup_{D \in \widehat{D}} D) = \widehat{Z} \cup \{v\}$, and
2. Each $z \in \widehat{Z}$ will have $\ell + 2$ private components $A_z^1, A_z^2, \dots A_z^{\ell+2} \in \widehat{D}$ such that $z \in N_G(A_z^i)$ for all $i \in [\ell + 2]$. By private we mean that the components $A_z^1, A_z^2, \dots A_z^{\ell+2}$ are all different for different $z \in \widehat{Z}$.

Lemma 8.30. *For any ℓ-pseudoforest deletion set of G that does not contain v, there exists an ℓ-pseudoforest deletion set X' in G such that $|X'| \leq |X|$, $X' \cap (\bigcup_{A \in \widehat{D}} A) = \emptyset$ and $\widehat{Z} \subseteq X'$.*

Proof. Let X be an ℓ-pseudoforest deletion set of G. We take $X' = (X \setminus \bigcup_{A \in \widehat{D}} A) \cup \widehat{Z}$. First we prove that $|X'| \leq |X|$. By definition of X' we just need to show that $|X \cap (\widehat{Z} \cup \bigcup_{A \in \widehat{D}} A)| \geq |\widehat{Z}|$. For each $z \in \widehat{Z}$, let us look at the graph induced on $Y_z = (\bigcup_{i \in [\ell+2]} A_z^i) \cup \{v, z\}$. Since each of the connected components A_z^i are trees for all $i \in [\ell + 2]$, there exists a unique path from v to z through each A_z^i. Hence, there are $\ell + 2$ vertex disjoint paths from v to z in $G[Y_z]$. This means that for any edge set F of size at most ℓ, deleting F will not make the connected component a tree (as two vertex disjoint paths will still remain, which would form a cycle) and hence $G[Y_z]$ is not an ℓ-pseudoforest. Let $Y = \bigcup_{z \in \widehat{Z}} Y_z$. In $G[Y]$, we have $|\widehat{Z}|$ graphs which are not ℓ-pseudoforests and are vertex disjoint except for the vertex v. But we have assumed that $v \notin X$, so all these $|\widehat{Z}|$ graphs contribute at least 1 to X. Hence X contains at least $|\widehat{Z}|$ vertices from $\widehat{Z} \cup \bigcup_{A \in \widehat{D}} A$.

To show that X' is an ℓ-pseudoforest deletion set for G, let $F = \bigcup_{A \in \widehat{D}} A$. We look at the graph $G - (X' \cup F)$ and call it G'. Since G' is a subgraph of $G - X$, it is an ℓ-pseudoforest. Observe that G' is same as $(G - X') - F$. $(G - X')[F]$ is a forest and each of the trees in $(G - X')[F]$ is connected to G' via a single edge (to v). So, we can keep applying Reduction Rule 16 on $G - X'$ to delete vertices of F to eventually get G', which is an ℓ-pseudoforest. Since we have preserved equivalence at each step of applying the reduction rule, $(G - X')$ is an ℓ-pseudoforest and X' is an ℓ-pseudoforest deletion set for G. $\square$

Now we are ready to give the final reduction rule.

Reduction Rule 22. *Delete all edges between v and $\bigcup_{A \in \widehat{D}} A$ and put $\ell + 2$ parallel edges between v and z for all $z \in \widehat{Z}$.*

Lemma 8.31. *Reduction Rule 22 is correct.*

Proof. Let (G, k) be a Yes instance of ℓ-pseudoforest Deletion and let X be an ℓ-pseudoforest deletion set of size at most k for G. If $v \in X$, then X is an ℓ-pseudoforest deletion set for G' also because $G - \{v\}$ is same as $G' - \{v\}$. Now, if $v \notin X$, then we can assume by Lemma 8.30 that $\widehat{Z} \in X$, but then again, $G - \widehat{Z}$ is same as $G' - \widehat{Z}$ and X remains to be an ℓ-pseudoforest deletion set.

For the converse, let (G', k) be a Yes instance and X be an ℓ-pseudoforest deletion set of size at most k for G'. Again, if $v \in X$, then $G - X$ is same as $G' - X$, and hence G is a Yes instance. Now, if $v \notin X$, then we know that $z \in X$ for all $z \in \widehat{Z}$, because we have added $\ell + 2$ parallel edges between v and z. So we have that $\widehat{Z} \in X$. Let $F = \bigcup_{A \in \widehat{D}} A$. Looking at $(G - (X \cup F))$, we know that it is a subgraph of $(G' - X)$ and hence is an ℓ-pseudoforest. We see that $G - (X \cup F)$ is same as $(G - X) - F$. We also observe that $(G - X)[F]$ is a forest such that every tree in it is connected to v via exactly one edge. So, we can keep applying Reduction Rule 16 on $G - X$ to delete vertices of F to eventually get $(G - X) - F$, which is an ℓ-pseudoforest. Since we have preserved equivalence at each step of applying the reduction rule, $G - X$ is also an ℓ-pseudoforest and X is an ℓ-pseudoforest deletion set for G. So we have that (G, k) is a YES instance of ℓ-PSEUDOFOREST DELETION. □

Now we are ready to prove the final theorem of the section.

Theorem 8.7. *Given an instance (G, k) of ℓ-PSEUDOFOREST DELETION, in polynomial time, we can get an equivalent instance (G', k') such that $k' \leq k$, $|V(G')| \leq |V(G)|$ and maximum degrees of G' is at most $(k + \ell)(3\ell + 8)$.*

Proof. If the maximum degree of the graph is at most $(k + \ell)(3\ell + 8)$, then we are already done. Otherwise (if the maximum degree is greater than $(k + \ell)(3\ell + 8)$), one of reduction rules 16-22 applies. It can be easily seen that none of the reduction rules increases k or $|V(G)|$.

Now we have to show that these reduction rules can be applied only polynomially many times. For that, we use the measure approach. We define a measure for the input graph which satisfies the following three properties.

1. It is polynomial in size of the graph initially.
2. It is always non-negative.
3. It decreases by a non-zero constant after application of each reduction rule if the reduction rule does not terminate the algorithm.

Let $E_{\leq \ell+2} \subseteq E(G)$ denote the set of edges of G with multiplicity at most $\ell + 2$. We define the measure for the graph G to be the following.

$$\phi(G) = 2|V(G)| + |E_{\leq \ell+2}|$$

Clearly, the measure is polynomial in the size of the graph G initially and remains non-negative throughout. Reduction rules 16, 18, 19 and 20 delete some vertex from the graph, and hence decrease $|V(G)|$ while not increasing $|E_{\leq \ell+2}|$. Reduction Rule 17, when applicable, reduces $|E_{\leq \ell+2}|$ by at least 1 while not changing $|V(G)|$. Reduction Rule 21 terminates the algorithm when applicable. Hence, the only reduction rule which remains to be examined is 22. While applying Reduction Rule 22, we delete a nonzero number of edges of multiplicity 1, and hence $|E_{\leq \ell+2}|$ and $\phi(G)$ decrease by at least 1. This concludes the proof of the theorem. $\square$

8.5.2 Kernels through Protrusions

In this section, we show that ℓ-PSEUDOFOREST DELETION admits polynomial kernels using the concept of protrusions once again. Now, we first prove the following lemma.

Lemma 8.32. *If (G, k) is a* YES *instance for ℓ-*PSEUDOFOREST DELETION *and the maximum degree of a vertex in G is bounded by d, then G has a $(4kd(\ell+2), 2(\ell+2))$-protrusion decomposition.*

Proof. Let X be the ℓ-pseudoforest deletion set of G of size at most k. Let $G' = G - X$. Since G' is an ℓ-pseudoforest, by Lemma 8.4 the treewidth of G' is bounded by $\ell + 1$. Since maximum degree of any vertex in G is bounded by d, we have that $|N[X]| \leq kd$. Hence, by Lemma 8.12, we have that G admits a $(4kd(\ell+2), 2(\ell+2))$-protrusion decomposition. $\square$

Theorem 8.8. *ℓ-*PSEUDOFOREST DELETION *admits a kernel with $c_\ell k^2$ vertices, where the constant c_ℓ is a function of ℓ alone.*

Proof. By Theorem 8.7, we know that given an instance (G, k) of ℓ-PSEUDOFOREST DELETION, in polynomial time, we can get an equivalent instance (G', k') such that $k' \leq k$, $|V(G')| \leq |V(G)|$ and maximum degrees of G' is at most $(k + \ell)(3\ell + 8)$. Hence, by Lemma 8.32, we have that G' admits an $(f(k, \ell), 2(\ell + 2))$-protrusion decomposition where $f(k, \ell) = 4k(\ell + 2)(k + \ell)(3\ell + 8) = \mathcal{O}(k\ell^2(k + \ell))$. Now, we put $r = 3(2\ell + 5)$. By Theorem 8.1 We know that ℓ-PSEUDOFOREST DELETION has a protrusion replacer which replaces r-protrusions of size at least r'. Let $s = r' \cdot 2^r$. Note that r', r and s depend only on ℓ and hence are a constant for any instance of ℓ-PSEUDOFOREST DELETION.

Applying Theorem 8.3 on (G', k') with the above mentioned value of r, s and $\alpha = f(k, \ell)$

and $\beta = 2(\ell + 2)$, we know that there is an algorithm that runs in time $\mathcal{O}(m + n)$ and produces an equivalent instance (G'', k'') with $|V(G'')| \leq |V(G')|$ and $k'' \leq k'$. Also, since G' has a $(f(k, \ell), 2(\ell + 2))$-protrusion decomposition, if $|V(G)| \geq \alpha \cdot 244s$, then we have that $|V(G'')| \leq (1 - \delta)|V(G')|$ for some constant δ. This means that $|V(G'')| < |V(G')|$.

Given an instance (G, k), the kernelization algorithm for ℓ-PSEUDOFOREST DELETION first applies Theorem 8.7 to bound the maximum degree and then apply LPR with above mentioned parameters. As shown above, the process either results in a an equivalent instance (G', k') with $|V(G')| < |V(G)|$ or we have that $|V(G)| < f(k, \ell) \cdot 244s$. If we end up getting an instance (G', k') such that $|V(G')| < |V(G)|$, then we call the kernelization algorithm recursively on (G', k'), otherwise (if $|V(G')| = |V(G)|$) we output (G, k) as the kernel. Since each iteration of the algorithm decreases the number of the vertices the graph by at least 1, the number of recursive calls are bounded by $V(G)$. Also, since each call takes polynomial time, the total time taken by the kernelization algorithm is polynomial. For the output graph G, we have that $|V(G)| < f(k, \ell) \cdot 244s \leq c_\ell k^2$, where c_ℓ is a constant entirely depending on ℓ, we have that ℓ-PSEUDOFOREST DELETION admits a kernel with $c_\ell k^2$ vertices. □

8.5.3 An explicit kernel for Pseudoforest Deletion

To arrive at an explicit kernel for PSEUDOFOREST DELETION, in addition to bounding the maximum degree of the graph from above, we also need to show that the minimum degree of the graph is at least 3. We have already stated Reduction Rule 16 which takes care of vertices of degree at most 1. So we only need to take care of vertices of degree 2. To that end, we make use of the following reduction rule.

Reduction Rule 23. *If there exists a vertex $v \in V(G)$ of degree two then delete v and put a new edge between its neighbours (even if they were already adjacent). If both of v's edges are to the same vertex, delete v and put a new self loop on the adjacent vertex (even if it has self loop(s) already).*

The proof of correctness of Reduction Rule 23 is very similar to proof of correctness of Reduction Rule 15. Just that in this case, we do not worry about the set S. Also, it is easy to see that after the application of Reduction Rules 16 and 23 exhaustively, the minimum degree of the graph is at least 3. We prove the following lemma which talks about pseudoforest deletion sets of graphs of bounded degree with minimum degree 3.

Lemma 8.33. *If a graph G has minimum degree at least 3, maximum degree at most d, and pseudoforest deletion set of size at most k, then it has at most $k(d+1)$ vertices and at most $2kd$ edges.*

Proof. Let X be a pseudoforest deletion set of G of size at most k. Let $Y = V(G) \setminus X$. We first observe the following by summing up the degrees of vertices of Y.

$$3|Y| \leq 2(|E(G[Y])| + |E(X,Y)|$$

But we know that $|E(G[Y])| \leq |Y|$ and $|E(X,Y)| \leq kd$. Putting these in the above inequality, we get the following.

$$\begin{aligned} 3|Y| &\leq 2|Y| + kd \\ |Y| &\leq kd \end{aligned}$$

For bounding $|V(G)|$, we have the following.

$$\begin{aligned} |V(G)| &\leq |X| + |Y| \\ |V(G)| &\leq k(d+1) \end{aligned}$$

For bounding the number of edges, since $|Y| \leq kd$, we know that $|E(G[Y])| \leq kd$. Now we can bound $|E[G]|$ as following.

$$\begin{aligned} |E[G]| &\leq |E(G[X])| + |E(X,Y)| + |E(G[Y])| \\ &\leq kd + kd \\ &= 2kd \end{aligned}$$

This concludes the proof of the lemma. $\square$

Theorem 8.9. Pseudoforest Deletion *admits a kernel with $\mathcal{O}(k^2)$ edges and $\mathcal{O}(k^2)$ vertices.*

Proof. We know from Theorem 8.7 that given an instance (G, k) of Pseudoforest Deletion, in polynomial time, we can get an equivalent instance (G', k') such that $k' \leq k$, $|V(G')| \leq |V(G)|$ and maximum degree of G is at most $11(k+1)$. Then we apply Reduction Rules 16 and 23 exhaustively. These reduction rules do not change the value of k and it is easy to see that they do not increase degree of any vertex. Also, since both of the reduction rules delete a vertex from the graph, they can only be applied $|V(G)|$ times. Hence, we have a polynomial time algorithm at the end of which the minimum degree of the graph is at least 3 while the maximum degree is at most $11(k+1)$. By Lemma 8.33, if G' has more than $k(11k+12)$ vertices or more than $22k(k+1)$ edges, then we can safely say No. Otherwise we have a kernel with $\mathcal{O}(k^2)$ edges and $\mathcal{O}(k^2)$ vertices. □

8.6 Conclusions

In this chapter we studied the problem of deleting vertices to get to a graph where each component is ℓ edges away from being a tree. We obtained uniform kernels as well as an FPT algorithm for the problem when parameterized by the solution size. We believe that it would be interesting to study more problems under this view of parameterization. In fact, in the next chapter, we study one such generalization of the well known Odd Cycle Transversal problem.

Chapter 9

Strong Graph Deletion: Bipartite Graphs

9.1 Introduction

Graph-modification by either deleting vertices or deleting edges or adding edges such that the resulting graph satisfies certain properties or becomes a member of some well-understood graph class is one of the basic problems in graph theory and graph algorithms. However, most of these problems are NP–complete [Yan78, LY80] and thus they are subjected to intensive study in algorithmic paradigms that are meant for coping with NP-completeness [Fuj98, LY94, FLMS12, MOR13]. These paradigms among others include applying restrictions on inputs, approximation algorithms and parameterized complexity. In the earlier chapters, we have seen some examples of such graph editing problems and algorithms for the same. The goal of this chapter is to introduce a 'stronger' notion of graph deletion in the realm of parameterized complexity and illustrate the difficulties that arise when considering the family of bipartite graphs and provide an approach to obtain fixed-parameter tractability by overcoming these difficulties.

A typical instance of parameterized graph deletion is of the following form. Let $\mathcal{F}$ be a family of graphs – such as edgeless graphs, forests, cluster graphs, chordal graphs, interval graphs, bipartite graphs, split graphs or planar graphs. The deletion problem corresponding to $\mathcal{F}$ is formally stated as follows.

$\mathcal{F}$-Vertex (Edge) Deletion **Parameter(s):** k

Input: An undirected graph G and a non-negative integer k.

Question: Does there exist $S \subseteq V(G)$ (or $S \subseteq E(G)$), such that $|S| \leq k$ and $G - S$ is in $\mathcal{F}$?

In other words, given a graph G, can we delete at most k vertices or k edges such that the resulting graph belongs to $\mathcal{F}$? An algorithm for $\mathcal{F}$-Vertex (Edge) Deletion that runs in time $f(k) \cdot |V(G)|^{\mathcal{O}(1)}$ is called *fixed-parameter tractable* (FPT) algorithm and the problem itself is said to be FPT.

The study of parameterized graph deletion problems together with their various restrictions and generalizations has been an extremely active sub-area over the last few years. In fact, just over the course of the last couple of years there have been results on parameterized algorithms for Chordal Editing [CM14], Unit Vertex (Edge) Deletion [Cao15], Interval Vertex (Edge) Deletion [CM15, Cao16], Planar $\mathcal{F}$ Deletion [FLMS12, KLP$^+$13], Planar Vertex Deletion [JLS14], Block Graph Deletion [KK15] and Simultaneous Feedback Vertex Set [ALMS16]. Several known parameterized algorithms for $\mathcal{F}$-Edge Deletion or the version where the objective is to delete k_1 vertices and k_2 edges utilize the fact that given a Yes-instance (G, k) or (G, k_1, k_2) to the problem, there exists a vertex set S^* of $V(G)$ of size $k = k_1 + k_2$ such that $G - S^*$ belongs to $\mathcal{F}$. Clearly, this is true for any hereditary family of graphs; that is, if $G \in \mathcal{F}$ then all its induced subgraphs belong to $\mathcal{F}$. Thus, if the corresponding $\mathcal{F}$-Vertex Deletion is FPT then we can apply this algorithm and find a vertex subset S^* of size at most k such that $G - S^* \in \mathcal{F}$. Having the set S^* allows one to infer numerous structural properties of the input which can then be utilized in non-trivial ways to solve the original $\mathcal{F}$-Vertex (Edge) Deletion problem. However, the existence of such a set is no longer guaranteed in the proposed stronger version of this problem.

Let $\mathcal{F}$ be a polynomial-time recognizable family of graphs; that is, given a graph G, in polynomial time we can decide whether G belongs to $\mathcal{F}$. For a fixed integer ℓ, let $\mathcal{F} + \ell e$ denote the class of graphs that can be obtained by adding ℓ edges to a graph in $\mathcal{F}$ [Cai03].

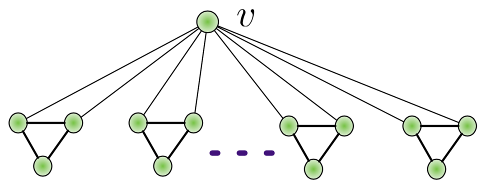


Figure 9.1: An example showing the strength of the new parameter for editing problems.

Furthermore, suppose that $\mathcal{F}$-Edge Deletion is FPT with running time $\mathcal{O}(g(\ell) \cdot n^c)$. Here, $n = |V(G)|$ and c is a fixed constant. That is, for any fixed integer ℓ, $\mathcal{F} + \ell e$ can be recognized in time $\mathcal{O}(n^c)$. The Strong $\mathcal{F}$-Deletion problem is defined as follows.

Strong $\mathcal{F}$-Deletion **Parameter(s):** k_1, k_2

Input: An undirected graph G and non-negative integers k_1 and k_2.

Question: Does there exist $S_1 \subseteq V(G)$ such that $|S_1| \leq k_1$ and every *connected component* of $G - S_1$ belongs to $\mathcal{F} + k_2e$?

A close inspection easily shows that Strong $\mathcal{F}$-Deletion is much stronger than $\mathcal{F}$-Deletion. For example, let $\mathcal{F}$ be the family of bipartite graphs and consider the graph G depicted in Figure 9.1. The graph G has $\frac{n-1}{3}$ vertex disjoint triangles and the vertex v is adjacent to two vertices from each of the triangles. Clearly, after deleting v every connected component can be made bipartite by deleting exactly one edge. Thus, in the traditional sense this is a yes-instance of Bipartite Deletion with parameters $(1, \frac{n-1}{3})$; however this is already a yes-instance of Strong Bipartite Deletion with parameters $(1, 1)$. Thus, these problems seem much harder than traditional editing problems as the parameters are much smaller.

An alternate viewpoint is that when G has a vertex set S of size at most k_1 such that every connected component of $G - S$ is in $\mathcal{F} + k_2e$ then S can be considered a *modulator* into the graph class where every connected component of the graph belongs to $\mathcal{F} + k_2e$. While modulators to various polynomial-time recognizable graph classes have been studied in a very systematic way [FJR13, FLMS12, Jan13], the same is not true of modulators to NP-complete graph classes. Studying the Strong $\mathcal{F}$-Deletion problem on the other hand allows us to do precisely this. In fact, it is not even necessary that the class $\mathcal{F}$ is a conventional graph class and instead can be the 'composition' of various graph classes. For instance $\mathcal{F}$ could be defined as the set of all graphs where each connected component is either chordal *or* a bipartite graph. The computation of such modulators is a problem with several algorithmic applications. For instance, in a recent work, Ganian et al. [GRS16] used a similar notion in order to design algorithms for the classic CONSTRAINT SATISFACTION PROBLEM.

In this chapter we study the STRONG $\mathcal{F}$-DELETION, when $\mathcal{F}$ is the family of bipartite graphs. Henceforth, $\mathcal{F}$ denotes the family of bipartite graphs. We call a graph G, t-*pseudobipartite*, if every connected component of G belongs to $\mathcal{F} + te$. The problem we study is as follows.

STRONG BIPARTITE DELETION **Parameter(s):** k, ℓ

Input: An undirected graph G and non-negative integers k and ℓ.

Question: Does there exist $S \subseteq V(G)$ such that $|S| \leq k$ and every connected component of $G - S$ belongs to $\mathcal{F} + \ell e$?

We refer to the set S as the *solution* for this instance. The primary reason behind the selection of the family of bipartite graphs for our study is that the problems where we are required to delete vertices and/or edges to obtain a bipartite graph are some of the most basic and well studied problems in parameterized complexity and studying these problems has led to the discovery of several new techniques and tools. These problems are called ODD CYCLE TRANSVERSAL and EDGE BIPARTIZATION in literature and the algorithms with best dependence on the parameter have running time $\mathcal{O}(2.3146^k n^{\mathcal{O}(1)})$ and $\mathcal{O}(1.977^k n^{\mathcal{O}(1)})$, respectively [LNR+14, PPW15].

In Chapter 8, we studied the problem of deleting k vertices to get to an ℓ-pseudoforest. Even though ℓ-PSEUDOFOREST DELETION and STRONG BIPARTITE DELETION look very similar in nature, we would like to draw the attention of the reader towards the key differences between the two problems. The algorithm in Chapter 8 for ℓ-PSEUDOFOREST DELETION runs in time $c_\ell^k n^{\mathcal{O}(1)}$ for every *fixed* ℓ, which is FPT time when parameterized by k alone for that fixed value of ℓ. If we make ℓ to be part of the input and a parameter, the we would need different protrusion replacers for different values of ℓ and it would not give a uniform algorithm for ℓ-PSEUDOFOREST DELETION. On the other hand, in STRONG BIPARTITE DELETION, we take both k and ℓ to be parts of the input and as parameters for the FPT algorithm, and give uniform FPT algorithms for the problem. Hence, the nature of results in this chapter is more general than that of the ones in Chapter 8.

Our Result and Methodology. Our main result is the following theorem.

Theorem 9.1. STRONG BIPARTITE DELETION *is* FPT.

The first big obstacle that needs to be overcome is the fact that the input graph can have a minimum odd cycle transversal of unbounded size. The first part of our algorithm is devoted to overcoming this obstacle. We utilise the technique of iterative compression to reduce it to a bounded number of equivalent instances each having an odd cycle transversal of size $poly(k, \ell)$. Then we use the recursive understanding technique introduced by Grohe et al. [GKMW11] (also see [CCH+12]) to first find a small separator in the graph which separates the graph into two parts, each of sufficiently large size and then recursively solve a 'border' version of the same problem on one of the two sides. The subroutines that we use to compute the separators are those of Chitnis et al.[CCH+12] who built upon the work of Kawarabayashi and Thorup [KT11]. The border version of the problem is a generalization which also incorporates a special bounded set of vertices, called terminals. During the course of our algorithm, we will attempt to solve the border problem on various subgraphs of the input graph. The objective in the border problem is to find a bounded set of vertices contained within a particular subgraph such that any vertex in this subgraph *not* in the computed set is not required in *any* solution for the given instance irrespective of the vertices chosen outside this subgraph.

Given the output of the border problem, the standard approach is to either delete the remaining vertices or simply 'bypass' these vertices. In our case, no such simple reduction seems likely. However, we show that by blowing up the size of the computed set by a function of the parameter, we can use a 'parity preserving' bypassing operation to get rid of the remaining vertices.

This leaves us with the base case of the recursion, that is when we are unable to find a small separator. At this stage, we know that the graph has a bounded sized odd cycle transversal *and* has a highly connected structure. Interestingly, even this seemingly significant structural information regarding the input does not seem enough to imply a straightforward algorithm. Instead, we compute an approximate oct solution and design a branching rule that has as its base case, the case when the approximate oct is not separated by the solution. Here, we rephrase this problem as a cut problem, namely MIXED MULTIWAY CUT-UNCUT (MMCU), which we study in detail in the next chapter. In the MMCU problem, the input is a multigraph G, integers k and ℓ, a set of terminals $T \subseteq V(G)$, and equivalence relation $\mathcal{R}$ on the set T. The objective is to output a solution $\mathcal{X} = (X, F)$ such that $X \subseteq (V(G) \setminus T)$, $F \subseteq E(G)$, $|X| \leq k$, $|F| \leq \ell$ and for all $u, v \in T$, u and v belong to the same connected component of $G - \mathcal{X}$ if and only if $(u, v) \in \mathcal{R}$ and $\perp$ if no such solution exists, , where $G - \mathcal{X}$ is the graph resulting from deleting X and F from G. The FPT algorithm for MMCU, which we defer to the next chapter, allows us to get our final result.

9.2 Preliminaries

In this section, we first give the notations and definitions which are used in this chapter. Then we state some known results which will be used later in the chapter.

Notations and Definitions: An *oct* of a graph G, is a set $X \subseteq V(G)$ such that $G - X$ is bipartite. Similarly, an *edge-oct* of a graph G is set $F \subseteq E(G)$ such that the graph $G - F$ is bipartite. We call a graph *ℓ-pseudobipartite*, if each of the connected components of G has an edge-oct of size at most ℓ. An *ℓ-pseudobipartite deletion set* of a graph G is a set $X \subseteq V(G)$ such that $G - X$ is ℓ-pseudobipartite.

Now we state the definitions of good node separations and flower separations from [CCH+12]. Then we state the lemmas that talk about the running time to find such separations and the properties of the graph if such separations do not exist.

Definition 9.2 (C.1 in [CCH+12])**.** *Let G be a connected graph and $V^\infty \subseteq V(G)$ a set of undeletable vertices. A triple (Z, V_1, V_2) of subsets of $V(G)$ is called a (q,k)-good node* *separation, if $|Z| \leq k$, $Z \cap V^\infty = \emptyset$, V_1 and V_2 are vertex sets of two different connected components of $G - Z$ and $|V_1 \setminus V^\infty|, |V_2 \setminus V^\infty| > q$.*

Definition 9.3 (C.2 in [CCH+12])**.** *Let G be a connected graph, $V^\infty \subseteq V(G)$ a set of undeletable vertices, and $T_b \subseteq V(G)$ a set of border terminals in G. A pair $(Z, (V_i)_{i=1}^r)$ is called a (q,k)-*flower separation* in G (with regard to border terminals T_b), if the following holds:*

- *$1 \leq |Z| \leq k$ and $Z \cap V^\infty = \emptyset$; the set Z is the core of the flower separation $(Z, (V_i)_{i=1}^r)$;*
- *V_i are vertex sets of pairwise different connected components of $G - Z$, each set V_i is a petal of the flower separation $(Z, (V_i)_{i=1}^r)$;*
- *$V(G) \setminus (Z \cup \bigcup_{i=1}^r V_i)$, called a stalk, contains more than q vertices of $V \setminus V^\infty$;*
- *for each petal V_i we have $V_i \cap T_b = \emptyset$, $|V_i \setminus V^\infty| \leq q$ and $N_G(V_i) = Z$;*
- *$|(\bigcup_{i=1}^r V_i) \setminus V^\infty| > q$.*

Lemma 9.1 (C.3 in [CCH+12])**.** *Given a connected graph G with undeletable vertices $V^\infty \subseteq V(G)$ and integers q and k, one may find in $\mathcal{O}(2^{\mathcal{O}(\min(q,k)\log(q+k))} n^3 \log n)$ time a (q,k)-good node separation of G, or correctly conclude that no such separation exists.*

Lemma 9.2 (C.4 in [CCH+12])**.** *Given a connected graph G with undeletable vertices $V^\infty \subseteq V(G)$ and border terminals $T_b \subseteq V(G)$ and integers q and k, one may find a (q,k)-flower separation in G w.r.t. T_b in $\mathcal{O}(2^{\mathcal{O}(\min(q,k)\log(q+k))} n^3 \log n)$ time, or correctly conclude that no such flower separation exists.*

Lemma 9.3 (C.5 in [CCH+12])**.** *If a connected graph G with undeletable vertices $V^\infty \subseteq V(G)$ and border terminals $T_b \subseteq V(G)$ does not contain a (q,k)-good node separation or a (q,k)-flower separation w.r.t. T_b then, for any $Z \subseteq V(G) \setminus V^\infty$ of size at most k, the graph $G - Z$ contains at most $(2q+2)(2^k - 1) + |T_b| + 1$ connected components containing a vertex of $V(G) \setminus V^\infty$, out of which at most one has more than q vertices not in V^∞.*

Other notation and definitions.

Definition 9.4. *Let G be a graph and A, B, S be a partition of $V(G)$. We say that (A, S, B) is a* separation *in G if $N(A), N(B) \subseteq S$.*

Definition 9.5. *Let G be a graph, X and Y be vertex sets. Let (X_1, X_2) and (Y_1, Y_2) be bipartitions of X and Y respectively. We say that (X_1, X_2) and (Y_1, Y_2) are compatible if the bipartition of $X \cap Y$ induced by both bipartitions are the same. If $Y \subseteq X$, we say that (X_1, X_2) extends (Y_1, Y_2) if $Y_1 \subseteq X_1$ and $Y_2 \subseteq X_2$.*

Definition 9.6. *Let G be a graph and $S \subseteq V(G)$. For every $v \in V(G) \setminus S$, we denote by $CC_{G-S}(v)$ the connected component of $G - S$ containing v. If X is a set of vertices in the same component of $G - S$ then we denote this component by $CC_{G-S}(X)$.*

9.3 Overview of the algorithm

To solve Strong Bipartite Deletion, we first reduce it to a slightly more general problem where we also have a set U of undeletable vertices and we are not allowed to select vertices in our potential solution from U. In particular the problem we will study is as follows.

Pseudobipartite Deletion **Parameter(s):** k, ℓ

Input: A graph G, integers k and ℓ and a set $U \subseteq V(G)$.

Question: Does there exist $X \subseteq V(G)$ such that $|X| \leq k$, $X \cap U = \emptyset$ and $G - X$ is ℓ-pseudobipartite?

Observe that when we set $U = \emptyset$ in Pseudobipartite Deletion, we get the Strong Bipartite Deletion problem. In rest of the chapter, we design an algorithm for Pseudobipartite Deletion using the method of iterative compression and recursive understanding technique introduced by Chitnis et al [CCH+12]. We describe the iterative compression phase of algorithm in Section 9.4.1, recursive phase of the algorithm in Section 10.5.3 and the high connectivity phase of the algorithm in Section 10.5.4. We start off by deleting the connected components from the graph which are already ℓ-pseudobipartite. Then using the technique of iterative compression, in Lemma 9.6, we reduce an instance of Pseudobipartite Deletion to $2^{\mathcal{O}(k)}$ many instances of the same problem such that the original instance is a Yes instance if and only if at least one of the new instances is a Yes instance. In addition to this, we also show that all the new instances have an oct of size bounded by a polynomial in k and ℓ. Then we take care of the case when the graph has more than one connected component. This can be easily done by solving the problem optimally on each connected component. Then we define the border problem, where we are additionally provided with some border terminals, say T.

Bordered-Pseudobipartite Deletion (B-PBD) **Parameter(s):** k,ℓ

Input: A Pseudobipartite Deletion instance $\mathcal{I} = (G, k, \ell, U)$ with G being connected and a set $T \subseteq V(G)$ such that $|T| \leq 2k$; denoted by $\mathcal{I}_b = (G, k, \ell, U, T)$.

Output: For each $\mathcal{P} \in \mathbb{P}(\mathcal{I}_b)$, output $\mathsf{sol}_{\mathcal{P}} = X_{\mathcal{P}}$ which is a *minimum* solution to $(\mathcal{I}_b, \mathcal{P})$, or find a *special* vertex v such that for each $\mathcal{P} \in \mathbb{P}(\mathcal{I}_b)$, there is a minimum solution $\mathsf{sol}^*_{\mathcal{P}}$ which contains v, otherwise output $\mathsf{sol}_{\mathcal{P}} = \perp$.

Here, the set $\mathbb{P}(\mathcal{I}_b)$ denotes the set of 'interactions' of the border terminals of instance $\mathcal{I}_b$ with a solution and the objective is to find a solution that corresponds to each possible interaction or to find a *special* vertex which is part of a solution for each possible interactions. It can be easily seen that for a Pseudobipartite Deletion instance $\mathcal{I} = (G, k, \ell.U)$, solving the B-PBD instance $(G, k, \ell.U, \emptyset)$ either gives a solution to the instance $\mathcal{I}$ or outputs a vertex which is part of a minimum solution for the instance $\mathcal{I}$, so in any case we make progress.

As is the case in algorithms based on the recursive understanding approach, to solve the border problem, we proceed to check whether a good separator T' exists in the graph. We use the notions of good node separations and good flower separations defined by Chitnis et al.[CCH+12] and look for good (q, k) node separation or (q, k) flower separation. The definitions are given in the preliminaries section. The running times required to compute such separations (if they exist) are argued by Lemmas 10.3 and 10.4. If we succeed in finding such a separation, then we see that we have divided the graph into two large parts using a small number of vertices. The definitions of node separations and flower separation help us argue that one of the parts (which contains at most half of the border terminals) is connected. We call the smaller graph H.

Now we update the set of terminals to include the separator, and solve the border problem $\mathcal{I}'_b$ recursively on the smaller graph. That is, for every behavior $\mathcal{P} \in \mathbb{P}(\mathcal{I}'_b)$ of the new border terminals, we get an optimum solution. Let Z denote the set $\bigcup_{\mathcal{P} \in \mathbb{P}(\mathcal{I}'_b)} \mathsf{sol}_{\mathcal{P}}$ where $\mathsf{sol}_{\mathcal{P}}$ is the solution for the smaller graph for behavior $\mathcal{P}$ of the border terminals. Then in Lemma 10.14, we argue that there exists a solution for the instance $\mathcal{I}_b$, which intersects with the smaller graph only in Z. At this time, we apply certain operations on the graph, such that the total number of the vertices in the graph G reduces by a sufficient amount. The approach of simply bypassing all the vertices not required by a solution does not quite work, as it could conceivably create spurious odd cycles which could lead to a Yes-instance turning into a No-instance. Hence, we make use of a 'parity preserving' bypassing operation to reduce the vertices of the graph, and show in Lemma 9.12 that this operation results in an equivalent instance.

Since the size of the set $\mathbb{P}(\mathcal{I}_b)$ can be bounded by some function of k and ℓ, and for each of them we need to keep some bounded number of vertices, we can guarantee that after the application of the recursive step (Step 8 in the algorithm) the number of vertices in the graph decreases by a sufficiently large number.

We then describe the algorithm for the problem in the case when there is no good-separation to recurse upon. This is done in Section 10.5.4. Here, we need to solve the Border-Pseudobipartite Deletion instance (G, k, ℓ, U, T) in the case that Step 8 is not applicable on the graph. For this, for each $\mathcal{P} \in \mathbb{P}(\mathcal{I}'_b)$, we solve the instance $(G_{\mathcal{P}}, k, \ell, U')$ as described in Section 10.5.3, where the graph $G_{\mathcal{P}}$ encoded the interaction, $\mathcal{P}$, of the border terminals T. We argue that in the absence of a good node separation or a flower separation of size at most k in the graph G, Lemma 10.5 guarantees an appropriate type of *high connectivity* in the graph G. In Lemma 9.14, we exploit this property to get a similar high connectivity property for the graph $G_{\mathcal{P}}$. Since the graph G has an oct of bounded size because of Lemma 9.6, we can show a similar bound on oct of the graph $G_{\mathcal{P}}$ also.

Once we are at this stage, we use the algorithm for finding oct given in [LNR$^+$14] to find an oct of bounded size and get an instance of the following problem.

OCT-PBD **Parameter(s):** k,ℓ

Input: An instance (G, k, ℓ, U) of Pseudobipartite Deletion along with an oct O of G of size at most $g(k, \ell)$ such that for any $Z \subseteq (V(G) \setminus U)$ of size at most k, in the graph $G - Z$, at most one connected component containing a vertex of $V(G) \setminus U$ has more than $h(k, \ell)$ vertices not in U.

Output: A minimum sized ℓ-pseudobipartite deletion set X of G of size at most k such that $X \cap U = \emptyset$. Output $\perp$ if such a set does not exist.

Then we guess the intersection of the oct with the solution. This lets us assume that the solution is disjoint from the oct. Formally, we branch into $2^{|O|}$ instances of the following problem.

OCT-PBD(I) **Parameter(s):** k,ℓ

Input: An instance (G, k, ℓ, U, O) of OCT-PBD.

Output: A minimum sized ℓ-pseudobipartite deletion set X of G of size at most k such that $X \cap (O \cup U) = \emptyset$. Output $\perp$ if such a set does not exist.

To solve an instance of OCT-PBD(I), we guess which vertices of the oct are going to be in the same connected component after deleting the solution. This gives us $g(k, \ell)^{g(k,\ell)}$ instances of following problem.

OCT-PBD(II) **Parameter(s):** k, ℓ

Input: An instance (G, k, ℓ, U, O) of OCT-PBD(I) and an equivalence relation $\S$ on O with the guarantee that there exists a minimum sized ℓ-pseudobipartite deletion set $X \subseteq V(G) \setminus (U \cup O)$ of G such that for all $u, v \in O$, u and v belong to the same connected component of $G - X$ if and only if $(u, v) \in \S$.

Output: A minimum sized ℓ-pseudobipartite deletion set X of G of size at most k such that $X \cap (O \cup U) = \emptyset$. Output $\perp$ if such a set does not exist.

In Lemma 9.15 we argue the correctness of this branching step. To solve an instance $(G, k, \ell, U, O, \S)$ of OCT-PBD(II), we look at the number of equivalence classes in $\S$. If it is more than one, then either we solve the problem via brute force on some connected component (of size at most $h(k, \ell)$) or we give a branching step where we solve at most

$h(k,\ell)+1$ instances OCT-PBD(II), where the size of the solution we are looking for has decreased by at least one. In the other case, that is when there is only one equivalence class in S, which means that all the vertices of oct are going to be part of a single connected component resulting by deleting a solution, we also guess how the oct vertices themselves will be bipartitioned eventually upon deleting the k vertices in the solution and an arbitrary but fixed set of at most ℓ edges which make the connected component containing these vertices, bipartite. So for each instance of OCT-PBD(II) , we get $2^{|O|}$ instances of following problem.

OCT-PBD(III) **Parameter(s):** k,ℓ

Input: An instance $(G,k,\ell,U,O,\S)$ of OCT-PBD(II) such that for all $u,v \in O$, $(u,v) \in \S$ and a bipartition $(O_1 \uplus O_2)$ of O with the guarantee that there exists a minimum sized ℓ-pseudobipartite deletion set $X' \subseteq V(G) \setminus (U \cup O)$ of G such that all vertices of O belong to the same connected component C of $G - X'$ and there exists an edge-oct F of $G[C]$ of size at most ℓ and a bipartition $(C_1 \uplus C_2)$ of C such that $G[C_1] - F$ and $G[C_2] - F$ are independent sets and $O_1 \subseteq C_1$ and $O_2 \subseteq C_2$.

Output: A minimum sized ℓ-pseudobipartite deletion set X of G of size at most k such that $X \cap (O \cup U) = \emptyset$. Output $\perp$ if such a set does not exist.

In Lemma 9.16, we argue that this branching is correct. Finally, to solve an instance $\mathcal{I}$ of OCT-PBD(III), we reduce it to an instance of MMCU* – a special case of mixed multiway cut-uncut (which we study in the next chapter) with some undeletable vertices.

MMCU* **Parameter(s):** k,ℓ

Input: A graph G, integers k and ℓ, $T \subseteq V(G)$, an equivalence relation $\mathcal{R}$ on T having at most two equivalence classes, and set of undeletable vertices $U \subseteq V(G)$.

Output: Output a minimal solution $\mathcal{X} = (X, F)$ to MMCU instance $(G, T, \mathcal{R}, k, \ell)$ such that $X \cap U = \emptyset$ and $\perp$ if no such solution exists.

Lemmas 9.17 and 9.18 argue the correctness of this reduction. Then in Theorem **??**, we state the the running time of MMCU* (the proof of this theorem is deferred to the next chapter), which allows us to show the desired FPT running time for Pseudobipartite Deletion.

9.4 An algorithm for Pseudobipartite Deletion

In this section, we describe the FPT algorithm for Pseudobipartite Deletion. We first prove the following lemma.

Lemma 9.4. *Given a connected graph G, we can recognize whether it is ℓ-pseudobipartite in $1.977^{\ell} n^{\mathcal{O}(1)}$ time.*

Proof. For a connected graph, the property of being ℓ-pseudobipartite is same as the property of the graph having an edge-oct of size at most ℓ, which can be checked in $1.977^{\ell} n^{\mathcal{O}(1)}$ time using algorithm in [PPW15]. $\square$

Now we give first step of the algorithm, where we get rid of connected components of the graph which are already ℓ-pseudobipartite.

Step 1. *Let (G, k, ℓ, U) be an instance of* Pseudobipartite Deletion. *Let $\mathcal{C}$ be the set of connected components of G. For each $C \in \mathcal{C}$, find if $G[C]$ is ℓ-pseudobipartite. Let $\mathcal{C}' = \{C \mid C \in \mathcal{C}$ and $G[C]$ is ℓ-pseudobipartite$\}$ and let $C' = \bigcup_{C \in \mathcal{C}'} C$. Pass the* Pseudobipartite Deletion *instance $(G - C', k, \ell, U \setminus C')$ to the next step.*

The correctness of the step follows from the fact that none of the vertices of any of the connected component deleted will be required by any solution of minimum size.

9.4.1 Preprocessing and bound on the size of an oct

In this section, with the help of the technique of iterative compression, given an instance of Pseudobipartite Deletion, we will reduce it to a bounded number of instances, such that they have an oct of bounded size and solving all of them also gives us a solution for the original instance.

Given an instance (G, k, ℓ, U) of Pseudobipartite Deletion, let $V(G) := \{v_1, v_2, \ldots, v_n\}$, $V_i := \{v_1, \ldots, v_i\}$ and the graph $G_i := G[V_i]$. We iterate through the instances $(G_i, k, \ell, U \cap V_i)$ starting from $i = k + 1$ and for the ith instance, with the help of a known solution S_i of size at most $k+1$ we try to find a solution S_i^* of size at most k. Formally, the compression problem we address is following.

PBD Compression (PBD-C) **Parameters:** k, ℓ
Input: A Pseudobipartite Deletion instance (G, k, ℓ, U) and an ℓ-pseudobipartite deletion set S of G of size at most $k+1$.
Question: Is there an ℓ-pseudobipartite deletion set of G of size at most k disjoint from U?

We solve the Pseudobipartite Deletion problem by generating and solving $n-k$ instances of the PBD-C problem as follows. Let $I_i = (G_i, k, \ell, U \cap V_i)$ be the ith instance of PBD-C. The set V_{k+1} is clearly an ℓ-pseudobipartite deletion set of size at most $k+1$ for G_{k+1}. It is also easy to see that if S_{i-1}^* is an ℓ-pseudobipartite deletion set of size at most k for the graph G_{i-1}, then the set $S_{i-1}^* \cup \{v_i\}$ is an ℓ-pseudobipartite deletion set of size at most $k+1$ for the graph G_i. We use these two observations to start off the iteration with the instance $(G_{k+1}, k, \ell, U, S_{k+1})$ of PBD-C where $S_{k+1} = V_{k+1}$ and check if G_{k+1} has an ℓ-pseudobipartite deletion set of size at most k disjoint from U. If there is such a solution S_{k+1}^*, we set $S_{k+2} := S_{k+1}^* \cup \{v_{k+2}\}$ and try to compute an ℓ-pseudobipartite deletion set of size at most k for the instance I_{k+2} and so on. If, during any iteration, the corresponding instance is found to be a No instance then it implies that the original instance is also a No instance. Finally the solution for the original input instance is the set S_n^*. Since there can be at most n such iterations, the total time taken is bounded by n times the time required to solve PBD-C.

Lemma 9.5. *Let (G, k, ℓ, U, S) be an instance of* PBD-C *such that none of the connected components of G are ℓ-pseudobipartite. If G has an ℓ-pseudobipartite deletion set disjoint from S of size at most k, then it has an oct of size at most $g'(k, \ell) := 2k + k\ell^2 + 1$.*

Proof. Let Z be an ℓ-pseudobipartite deletion set of G of size at most k disjoint from S. We know that every connected component of $G - Z$ is ℓ-pseudobipartite. We claim that there are at most $k + k\ell + 1$ many connected components of $G - Z$ which are not bipartite. To show this, let us observe that since none of the connected components of G is already ℓ-pseudobipartite, every connected component of $G - Z$ has a neighbour in

Z. Now, let assume for the sake of contradiction that there are more than $k + k\ell + 1$ connected components in $G - Z$ which are not bipartite. Out of these, there are at most $k + 1$ components such that they contain at least one vertex of S. So, there are more than $k\ell$ connected components disjoint from S which are not bipartite and have a neighbour in Z. By Pigeon Hole Principle, there exists $v \in Z$, such that v has a neighbour in at least $\ell + 1$ of these connected components. But then the graph induced on v along with these connected components is an induced subgraph of $G - S$, which is not ℓ-pseudobipartite. This is a contradiction to the fact that S was an ℓ-pseudobipartite deletion set of G.

Now to show an oct of bounded size, let $E_1, E_2, \ldots, E_r$ denote minimum sized edge-octs of connected components of $G - Z$ which are not bipartite and which do not contain any vertex of S. Let $E' = \cup_{1 \leq i \leq r} E_i$, and let $V_{E'}$ be the set formed by picking one endpoint for each edge in E' arbitrarily. Let the set O be defined as $O = S \cup Z \cup V_{E'}$. It is easy to see that O is an oct of G_Y. Since we know that $r \leq k\ell$, and $|E_i| \leq \ell$ for each $1 \leq i \leq r$, we have that $|V_{E'}| \leq k\ell^2$ and hence we have $|O| \leq 2k + k\ell^2 + 1$. $\square$

Now we prove the key lemma of the section which gives us a bound on the size of the oct of the graph.

Lemma 9.6. *There is an algorithm, which given an instance (G, k, ℓ, U, S) of* PBD-C*, runs in time $2^{\mathcal{O}(k\ell^2)} n^{\mathcal{O}(1)}$ and returns at most 2^{k+1} instances $\{I_1, I_2, \ldots, I_q\}$ of* Pseudobipartite Deletion*, where $I_i = (G_i, k_i, \ell, U)$ and $q \leq 2^{k+1}$ such that the following holds.*

- *(G, k, ℓ, U, S) is a* Yes *instance of* PBD-C *if and only if there is an $1 \leq i \leq q$ such that (G_i, k_i, ℓ, U) is a* Yes *instance of* Pseudobipartite Deletion.
- *For or each $1 \leq i \leq q$, $k_i \leq k$ and G_i has an oct of size at most $2k + k\ell^2 + 1$.*

Proof. Given an instance (G, k, ℓ, U, S) of PBD-C, for each $S' \subseteq (S \setminus U)$, we obtain an instance $I_{S'} := (G_{S'}, k_{S'}, \ell, U_{S'})$ by deleting S' from the graph G and, and applying Step 1 on the instance $(G - S', k - |S'|, \ell, U)$. Thus, we obtain at most 2^{k+1} instances of Pseudobipartite Deletion, each corresponding to a subset S' of $S \setminus U$.

- Let (G, k, ℓ, U, S) be a YES instance of PBD-C and let S^* be an ℓ-pseudobipartite deletion set of G disjoint from U of size at most k. Let $Y := S^* \cap S$. Let us now consider the PSEUDOBIPARTITE DELETION instance $(G - Y, k - |Y|, \ell, U)$. This instance is a YES instance of PSEUDOBIPARTITE DELETION, because $G - Y$ has a $k - |Y|$ sized ℓ-pseudobipartite deletion set, namely $S^* \setminus Y$. Then we apply Step 1 on this instance exhaustively, and the correctness of Step 1 implies that $I_Y = (G_Y, k_Y, \ell, U_Y)$ is a YES instance of PSEUDOBIPARTITE DELETION. We also observe that G_Y has an ℓ-pseudobipartite deletion set of size at most k_Y that is disjoint from S also, in addition to being disjoint from U.

  For the converse, it is easy to see that if $I_{S'}$ is a YES instance for PSEUDOBIPARTITE DELETION, then its solution when combined with the set S' gives a solution of size at most k for (G, k, ℓ, U, S), making it a YES instance as well.

- Clearly, for every $1 \leq i \leq q$, we have that $k_i \leq k$. Now, to see the bound on the size of an oct, we observe the following. If (G, k, ℓ, U, S) is a YES instance because of a solution $\hat{S}$ of size at most k and let $Y := S \cap \hat{S}$. So, as argued earlier, the instance $(G - Y, k - |Y|, \ell, U)$ has a solution which is disjoint from $S \setminus Y$. Since we only delete ℓ-pseudobipartite components of $G - Y$ to get to G_Y, (G_Y, k_Y, ℓ, U_Y) also has a solution disjoint from $S \setminus Y$. Hence, by Lemma 9.5, we know that G has an oct of size at most $g'(k, \ell)$. So, we can discard all the instances which do not have an oct of size at most $g'(k, \ell)$. The algorithm returns only the instances which have an oct of size at most $g'(k, \ell)$ and returns NO if no such instance exists.

To show the running time of the algorithm, we first observe that for each $Y \subseteq (S \setminus U)$, $(G - Y, k - |Y|, \ell, U)$ can be generated in polynomial time, and there are $2^{\mathcal{O}(k)}$ such instances. Now, for each of these instances, deleting the ℓ-pseudobipartite components using Lemma 9.4 can be done in $1.977^\ell n^{\mathcal{O}(1)}$ time and we get the instance I_Y. Then for each instance I_Y, we use the algorithm of [LNR+14] to check if the graph G_Y has an oct of size at most $g'(k, \ell)$ and discard it if it does not. Since $g'(k, \ell) = \mathcal{O}(k\ell^2)$, this algorithm runs in time $2.3146^{\mathcal{O}(k\ell^2)} n^{\mathcal{O}(1)}$. Hence the running time of the whole algorithm is bounded by $2^{\mathcal{O}(k\ell^2)} n^{\mathcal{O}(1)}$. This completes the proof of the lemma. $\square$

Henceforth, we will assume that the given Pseudobipartite Deletion instance is returned by algorithm of Lemma 9.6 and hence has an oct of bounded size.

9.4.2 Borders and Recursive Understanding

After applying Lemma 9.6, we still need to solve $2^{\mathcal{O}(k)}$ Pseudobipartite Deletion instances. In the this section and the next, we give an algorithm to solve instances which are given by Lemma 9.6. In fact, we will give an algorithm, which in addition to giving a Yes/No answer for these instances, also gives a minimum sized solution. In this section, we define the border problem and describe the recursive phase of the algorithm. We first state the step which will help us assume that the input graph is connected.

Step 2. *Let (G, k, ℓ, U) be an instance of* Pseudobipartite Deletion. *If G has more than k connected components which are not ℓ-pseudobipartite, then return* No. *Otherwise, solve the problem on each of the connected components individually to find the minimum set of vertices to be deleted to make them ℓ-pseudobipartite. If the summation of the sizes of individual solutions is larger than k, say* No. *Otherwise return the union of the solutions.*

The correctness follows from the fact that deletion of any k vertices will leave a connected component which is not ℓ-pseudobipartite untouched. Hence, now onwards, we will assume the graphs to be connected and to have an oct of size at most $2k + k\ell^2 + 1$. Now we describe the border problem.

Let $\mathcal{I} = (G, k, \ell, U)$ be a Pseudobipartite Deletion instance. The input to the border problem consists of an instance $\mathcal{I} = (G, k, \ell, U)$ of Pseudobipartite Deletion along with a set T of at most $2k$ vertices of G disjoint from U. The output to the border problem either consists of *several* solutions, one for each relevant 'behaviour' defined on the set of terminals, or it consists of a single *special* vertex, which is part of some minimum solution for every behaviour. In what follows, we will formalize this statement.

Let $\mathcal{I} = (G, k, \ell, U)$ be an instance of Pseudobipartite Deletion and $T \subseteq V(G)$. We let $\mathbb{P}(\mathcal{I}_b)$ denote the set of all tuples $\mathcal{P} = (X_T, \mathcal{R}, \mathcal{B}, L)$, such that $X_T \subseteq T$, $\mathcal{R}$ is an equivalence relation on $T \setminus X_T$, $\mathcal{B}$ is a bipartition of $T \setminus X_T$ and $L = \{(R_1, \ell_1), (R_2, \ell_2,) \dots (R_q, \ell_q)\}$ is a set of pairs which associates an integer $\ell_i \leq \ell$ with each equivalence class R_i in $\mathcal{R}$. For a tuple $\mathcal{P} = (X_T, \mathcal{R}, \mathcal{B}, L)$ with the equivalence classes of $\mathcal{R}$ being $R_1, \ldots, R_s$, the bipartition induced on the class R_i by $\mathcal{B}$ is denoted by (R_{i_1}, R_{i_2}).

For each $\mathcal{P} \in \mathbb{P}(\mathcal{I}_b)$, we define a super-graph $G_{\mathcal{P}}$ of G with the following additional vertices and edges.

- For each equivalence class R_i of $\mathcal{R}$, we add sets of vertices R'_{i_1} and R'_{i_2} such that $|R'_{i_1}| = |R'_{i_2}| = \ell + 1$. Then we add all the edges between vertices u and v such that $u \in R_{i_1} \cup R'_{i_1}$ and $v \in R_{i_2} \cup R'_{i_2}$. If $(R_i, \ell_i) \in L$, then we pick an arbitrary vertex $u^i \in R_i$ and add ℓ_i-many edge disjoint triangles which only intersect in u^i. That is, we add $2\ell_i$ new vertices $u_1^{i,a}, \ldots, u_{\ell_i}^{i,a}, u_1^{i,b}, \ldots, u_{\ell_i}^{i,b}$ and edges $\{(u^i, u_j^{i,a}), (u^i, u_j^{i,b}), (u_j^{i,a}, u_j^{i,b})\}$ for each $1 \leq j \leq \ell_i$.
- For each vertex $u \in X_T$, we add $\ell+1$-many edge disjoint triangles which only intersect in u. That is, we add $2(\ell + 1)$ new vertices $u_1^a, \ldots, u_r^a, u_1^b, \ldots, u_r^b$ where $r = \ell + 1$ and edges $\{(u, u_j^a), (u, u_j^b), (u_j^a, u_j^b)\}$ for each $1 \leq j \leq r$.

This completes the description of the graph $G_{\mathcal{P}}$. It can be seen that $|V(G_{\mathcal{P}}) \setminus V(G)| \leq 2k(4\ell + 1)$ and $|E(G_{\mathcal{P}}) \setminus E(G)| \leq 4k(2k + 3)(\ell + 1)$. The intuition behind the newly added vertices and edges is the following. Consider an instance of Pseudobipartite Deletion and suppose that G is a subgraph of the input instance with the terminal set T separating this subgraph from the rest of the input graph. The tuple $\mathcal{P} = (X_T, \mathcal{R}, \mathcal{B}, L)$ essentially captures the interaction of the terminal set with the solution and the rest of the graph. That is, X_T denotes the intersection of the solution with T. The partition $\mathcal{R}$ denotes the way the remaining vertices of T are partitioned as connected components after deleting a solution. The bipartition $\mathcal{B}$ denotes the bipartition of T induced by an arbitrary bipartition of the graph obtained from the input graph by deleting the k vertices in the solution and then a minimum edge-oct in the rest of the graph. Finally, for each $R \in \mathcal{R}$, if $(R, x) \in L$, it means that after deleting the k vertices in a solution, there is a set of at most x edges in the connected component containing R such that they are part of a minimum edge-oct of this component *and* they lie outside G. The newly added vertices essentially simulate this interaction of the terminals with the vertices in the solution. This will be proved formally in Lemma 9.8.

We denote by $U_{\mathcal{P}}$ the union of the set U with the vertices in $V(G_{\mathcal{P}}) \setminus V(G)$. Also, for $\mathcal{P} = (X_T, \mathcal{R}, \mathcal{B}, L)$, for each $R_i \in \mathcal{R}$, we denote by $\mathrm{H}^G_{\mathcal{P}}(R_i)$ the set of vertices in $V(G_{\mathcal{P}}) \setminus V(G)$ which are adjacent to a vertex of R_i. We drop the reference to G if it is clear from the context.

We say that a set $X \subseteq V(G) \setminus U$ is a *solution* to $(\mathcal{I}_b, \mathcal{P})$ where $\mathcal{P} = (X_T, \mathcal{R}, \mathcal{B}, L) \in \mathbb{P}(\mathcal{I}_b)$ if X is a solution to the Pseudobipartite Deletion instance $(G_{\mathcal{P}}, k, \ell, U'_{\mathcal{P}})$ where $U'_{\mathcal{P}} = U_{\mathcal{P}} \cup (T \setminus X_T)$. Recall that when we say that a set S is a solution to a Pseudobipartite Deletion instance (G, k, ℓ, U), we mean that $|S| \leq k$, $S \cap U = \emptyset$ and $G - S$ is ℓ-pseudobipartite. We are now ready to restate the formal definition of the border problem.

Bordered-Pseudobipartite Deletion (B-PBD) **Parameter(s):** k,ℓ
Input: A Pseudobipartite Deletion instance $\mathcal{I} = (G, k, \ell, U)$ with G being connected and a set $T \subseteq V(G)$ such that $|T| \leq 2k$; denoted by $\mathcal{I}_b = (G, k, \ell, U, T)$.
Output: For each $\mathcal{P} \in \mathbb{P}(\mathcal{I}_b)$, output $\mathsf{sol}_{\mathcal{P}} = X_{\mathcal{P}}$ which is a *minimum* solution to $(\mathcal{I}_b, \mathcal{P})$, or find a *special* vertex v such that for each $\mathcal{P} \in \mathbb{P}(\mathcal{I}_b)$, there is a minimum solution $\mathsf{sol}^*_{\mathcal{P}}$ which contains v, otherwise output $\mathsf{sol}_{\mathcal{P}} = \bot$.

Now we state a simple observation which tells us how to use the algorithm for Border-Pseudobipartite Deletion to solve an instance of Pseudobipartite Deletion.

Observation 9.7. *Let $\mathcal{I} = (G, k, \ell, U)$ be a* Pseudobipartite Deletion *instance with G being connected. Then the correct solution for the B-PBD instance $\mathcal{I}_b := (G, k, \ell, U, \emptyset)$ corresponding to $\mathcal{P} \in \mathbb{P}(\mathcal{I}_b)$ where $\mathcal{P} = (\emptyset, \emptyset, \emptyset, (\emptyset, 0))$ is a minimum sized solution for $\mathcal{I}$. Also, if the algorithm for B-PBD instance $\mathcal{I}_b$ returns a special vertex v, then there exists a minimum solution for the instance $\mathcal{I}$ containing v.*

In what follows we will show that *given* a correct output for an instance of B-PBD, there is an FPT algorithm which either computes an equivalent instance whose size is bounded by a function of k and ℓ or it outputs a special vertex as described in the problem definition. We begin with the following lemma which describes the interaction of the solution with the new vertices added during the construction of the graph $G_{\mathcal{P}}$.

Lemma 9.8. *Let $\mathcal{I}_b = (G, k, \ell, U, T)$ be an instance of* B-PBD. *Let $\mathcal{P} = (X_T, \mathcal{R}, \mathcal{B}, L) \in \mathbb{P}(\mathcal{I}_b)$ and let X be an arbitrary solution for $(\mathcal{I}_b, \mathcal{P})$. Then, the following properties hold.*

1. *X is disjoint from $T \setminus X_T$.*
2. *$X \supseteq X_T$*
3. *For each $R_i \in \mathcal{R}$, the vertices in R_i are in the same connected component of $G_{\mathcal{P}} - X$, denoted by C_i.*
4. *For each $R_i \in \mathcal{R}$, any minimum edge-oct for the component C_i contains exactly one edge from the set $\{(u^i, u_j^{i,a}), (u^i, u_j^{i,b}), (u_j^{i,a}, u_j^{i,b})\}$ for each $1 \leq j \leq \ell_i$.*

Proof. For the first statement, since X is a solution for the Pseudobipartite Deletion instance $(G_{\mathcal{P}}, k, \ell, U'_{\mathcal{P}})$, by definition it must be disjoint from $T \setminus X_T$.

For the second statement, suppose that there is a vertex $u \in X_T \setminus X$. Recall that $G_{\mathcal{P}}$ contains $\ell + 1$ edge-disjoint triangles one for each $1 \leq j \leq \ell + 1$, formed by the edge set $\{(u, u_j^a), (u, u_j^b), (u_j^a, u_j^b)\}$. Since $u \notin X$ and all the triangle vertices are in $U'_{\mathcal{P}}$, there is a connected component of $G_{\mathcal{P}} - X$ which contains all of these triangles. Since any such component is not ℓ-pseudobipartite, we obtain a contradiction to our assumption that X is a solution for $(\mathcal{I}_b, \mathcal{P})$.

The third statement simply follows from the fact that X is disjoint from $T \setminus X_T$ (as we have already argued) and it is also disjoint from the vertices in $R'_{i_1} \cup R'_{i_2}$ since these are contained in $U'_{\mathcal{P}}$.

Part of the argument for the final statement is analogous to that for the second statement. That is, since the vertex u^i is not contained in X, any edge-oct for the connected component containing u^i contains *at least* one edge from each of the triangles $\{(u^i, u_j^{i,a}), (u, u_j^{i,b}), (u_j^{i,a}, u_j^{i,b})\}$ where $1 \leq j \leq \ell_i$. The fact that a *minimum* edge-oct contains exactly one edge from each of these triangles follows from the fact that picking an arbitary edge from each of these triangles is indeed sufficient to make the graph induced on u_i along with these vertices, bipartite. This completes the proof of the lemma. $\square$

Next we give an observation and a lemma that allows us to merge the solution after doing the recursive step of our algorithm.

Observation 9.9. *Let G be a graph and (A, S, B) be a separation in G. Let (S_1, S_2) be a bipartition of S such that $G[A \cup S]$ is bipartite with a bipartition extending (S_1, S_2) and $G[B \cup S]$ is bipartite with a bipartition extending (S_1, S_2). Then, G is bipartite with a bipartition extending (S_1, S_2).*

Lemma 9.10. *Let $\mathcal{I}_b = (G, k, \ell, U, T)$ be an instance of* B-PBD*. Let $T' \subseteq V(G) \setminus U$ and let C be a connected component of $G - T'$ such that $N(C) = T'$. Let $H = G[C \cup T']$ and let $Q = T' \cup (C \cap T)$ and suppose that $|Q| \leq 2k$. Let $\mathcal{I}'_b$ denote the* B-PBD *instance $(H, k, \ell, (U \cap V(H)), Q)$. Let Z denote the set $\bigcup_{\mathcal{P} \in \mathbb{P}(\mathcal{I}'_b)} \mathsf{sol}_{\mathcal{P}}$. Then, for each $\mathcal{P} \in \mathbb{P}(\mathcal{I}_b)$, if there is a solution for $(\mathcal{I}_b, \mathcal{P})$, then there is one whose intersection with C is in Z. Moreover, if there exists a vertex v such that $v \in \mathsf{sol}_{\mathcal{P}'}$ for all $\mathcal{P}' \in \mathbb{P}(\mathcal{I}_b)$, then for all $\mathcal{P} \in \mathbb{P}(\mathcal{I}'_b)$, there exists a minimum solution to $(\mathcal{I}_b, \mathcal{P})$ which contains v.*

Proof. Fix a tuple $\mathcal{P} = (X_T, \mathcal{R}, \mathcal{B}, L) \in \mathbb{P}(\mathcal{I}_b)$ and a solution $X_{\mathcal{P}}$ for $(\mathcal{I}_b, \mathcal{P})$. Let Y_1 denote the set $X_{\mathcal{P}} \cap V(H)$. If $Y_1 \subseteq Z$, then we are done. Suppose that this is not the case. We now define a tuple $\hat{\mathcal{P}} = (\hat{X}_T, \hat{\mathcal{R}}, \hat{\mathcal{B}}, \hat{L}) \in \mathbb{P}(\mathcal{I}'_b)$ as follows.

1. $\hat{X}_T = X_{\mathcal{P}} \cap Q$

2. $\hat{\mathcal{R}} = \{R | R \subseteq (Q \setminus \hat{X}_T) \text{ such that } \forall u, v \in R, CC_{G_{\mathcal{P}} - X_{\mathcal{P}}}(u) = CC_{G_{\mathcal{P}} - X_{\mathcal{P}}}(v)\}$. That is, the equivalence classes in $\hat{R}$ are precisely those subsets of $Q \setminus \hat{X}_T$ which lie in the same connected component of $G_{\mathcal{P}} - X_{\mathcal{P}}$.

3. $\hat{\mathcal{B}} = (Q_1, Q_2)$ where $Q_1 \uplus Q_2 = Q \setminus \hat{X}_T$, (Q_1, Q_2) is compatible with $\mathcal{B}$ and for some minimum edge-oct $E_{\mathcal{P}}$ for $G_{\mathcal{P}} - X_{\mathcal{P}}$, there is a valid bipartition of the graph $G_{\mathcal{P}} - X_{\mathcal{P}} - E_{\mathcal{P}}$ which extends (Q_1, Q_2). For each $\hat{R}_i \in \hat{\mathcal{R}}$, we denote by $(\hat{R}_{i_1}, \hat{R}_{i_2})$ the bipartition of $\hat{R}_i$ induced by $\hat{B}$.

4. For each $\hat{R}_i \in \hat{\mathcal{R}}$ such that $\hat{R}_i \subseteq C$, it must be the case that $\hat{R}_i \subseteq T \setminus T'$ and $\hat{R}_i = R_j \in \mathcal{R}$ for some j. Therefore, L contains the pair $(\hat{R}_j, x)$ for some $x \leq \ell$.

Hence, for each $\hat{R}_i \in \hat{\mathcal{R}}$ such that $\hat{R}_i \subseteq C$, we add to $\hat{L}$ the pair (R_i, x).

Now, consider $\hat{R}_i \in \hat{\mathcal{R}}$ such that $\hat{R}_i \cap T = \emptyset$ (that is, $\hat{R}_i \subseteq T' \setminus T$). Observe that by definition of $\hat{R}$, the vertices in $\hat{R}_i$ are contained in the same connected component of $G_{\mathcal{P}} - X_{\mathcal{P}}$ and this component, denoted by $CC_{G_{\mathcal{P}}-X_{\mathcal{P}}}(\hat{R}_i)$ is disjoint from any vertex of T (by our assumption that $\hat{R}_i \cap T = \emptyset$). Now, by the definition of $\hat{B}$, we have a bipartition of $\hat{R}_i$, say $(\hat{R}_{i_1}, \hat{R}_{i_2})$ which can be extended by some bipartition of $CC_{G_{\mathcal{P}}-X_{\mathcal{P}}}(\hat{R}_i) - E'$ where E' is an edge-oct of $CC_{G_{\mathcal{P}}-X_{\mathcal{P}}}(\hat{R}_i)$ of size at most ℓ. Let x denote the number of edges in E' with at least one endpoint in $V(G) \setminus V(H)$. We add to $\hat{L}$ the pair $(\hat{R}_i, x)$.

Finally, for every other $\hat{R}_i \in \hat{\mathcal{R}}$, there must be an $R_j \in \mathcal{R}$ such that $\hat{R}_i \supseteq (R_j \cap V(H))$ (by definition of $\hat{\mathcal{R}}$). Consider the connected component $C' = CC_{G_{\mathcal{P}}-X_{\mathcal{P}}}(\hat{R}_i)$. By definition of $\mathcal{B}$ and $\hat{\mathcal{B}}$, there is a set E' of at most ℓ edges in C' such that $C' - E'$ is a bipartite graph with a bipartition simultaneously extending *both* (R_{j_1}, R_{j_2}) and $(\hat{R}_{i_1}, \hat{R}_{i_2})$. Let y be the number of edges of E' with both endpoints in the set $V(H)$. We set $x = \ell - y$ and add to $\hat{L}$ the pair $(\hat{R}_i, x)$.

This completes the description of the tuple $\hat{\mathcal{P}}$. Let $\hat{\mathsf{sol}}_{\hat{\mathcal{P}}}$ denote a solution to $(\mathcal{I}'_b, \hat{\mathcal{P}})$. It follows from the definition of $\hat{\mathcal{P}}$ that $X_{\mathcal{P}} \cap V(H)$ is in fact a solution to $(\mathcal{I}'_b, \hat{\mathcal{P}})$ and hence $\hat{\mathsf{sol}}_{\hat{\mathcal{P}}}$ exists. However, we now claim that $Y_{\mathcal{P}} = (X_{\mathcal{P}} \setminus V(H)) \cup \hat{\mathsf{sol}}_{\hat{\mathcal{P}}}$ is also solution for $(\mathcal{I}_b, \mathcal{P})$.

Observe that since $\hat{\mathsf{sol}}_{\hat{\mathcal{P}}}$ is a *minimum* solution, the set $Y_{\mathcal{P}}$ is no larger than $X_{\mathcal{P}}$. Therefore, it suffices to prove that $G_{\mathcal{P}} - Y_{\mathcal{P}}$ is ℓ-pseudobipartite. If this were not the case, then there is a connected component J in $G_{\mathcal{P}} - Y_{\mathcal{P}}$ which does not have an edge-oct of size at most ℓ. If J is disjoint from $X_{\mathcal{P}}$, then there is a connected component of $G_{\mathcal{P}} - X_{\mathcal{P}}$ which contains J. Since $G_{\mathcal{P}} - X_{\mathcal{P}}$ is ℓ-pseudobipartite, we contradict our assumption that the graph induced on J is not. Hence, we conclude that J must intersect $X_{\mathcal{P}}$. Similarly, if J is disjoint from $V(H)$ (but intersects $X_{\mathcal{P}} \setminus V(H)$ which is contained in $Y_{\mathcal{P}}$), then J cannot be disjoint from $Y_{\mathcal{P}}$, a contradiction. Henceforth, we assume that J intersects $X_{\mathcal{P}} \cap V(H)$. However, observe that if J is disjoint from Q, then it is also present in $H_{\hat{\mathcal{P}}} - \hat{\mathsf{sol}}_{\hat{\mathcal{P}}}$, contradicting our assumption that $\hat{\mathsf{sol}}_{\hat{\mathcal{P}}}$ is an ℓ- pseudobipartite deletion set for $H_{\hat{\mathcal{P}}}$. Hence, it must be the case that J intersects Q. We now consider the following 3 cases.

Case 1: $J \cap Q \subseteq T \setminus T'$. By the definition of $\hat{\mathcal{R}}$, there is exactly one equivalence class $\hat{R}_i \in \hat{\mathcal{R}}$ which intersects J, $\hat{R}_i = J \cap Q = J_1$ and moreover, there is an index j such that $\hat{R}_i = R_j \in \mathcal{R}$. Also, the bipartition $(\hat{R}_{i_1}, \hat{R}_{i_2})$ is same as the bipartition (R_{j_1}, R_{j_2}).

Let x be such that $(R_j, x) \in L$. Recall that by the definition of $\hat{L}$, the pair $(\hat{R}_i, x) \in \hat{L}$. Furthermore, since $\hat{\mathsf{sol}}_{\hat{\mathcal{P}}}$ is a solution for $(\mathcal{I}'_b, \hat{\mathcal{P}})$, the connected component $CC_{H_{\hat{\mathcal{P}}} - \hat{\mathsf{sol}}_{\hat{\mathcal{P}}}}(\hat{R}_i)$ has an edge-oct of size at most ℓ out of which exactly x edges have at least one endpoint in $\mathrm{H}_{\hat{\mathcal{P}}}(\hat{R}_i)$ (Lemma 9.8). Therefore, there is a set of at most $\ell - x$ edges in $H - \hat{\mathsf{sol}}_{\hat{\mathcal{P}}}$ whose deletion from the graph induced on $J - \mathrm{H}_{\hat{\mathcal{P}}}(\hat{R}_i) = J - \mathrm{H}_{\mathcal{P}}(R_j)$ results in $(\hat{R}_{i_1}, \hat{R}_{i_2})$ being a valid bipartition. Combining these edges with the exactly x edges with an endpoint in $\mathrm{H}_{\mathcal{P}}(R_j)$ given by Lemma 9.8, we obtain an edge-oct of size at most ℓ for the connected component J, a contradiction. This completes the argument for the first case.

Case 2: $J \cap Q \subseteq T' \setminus T$. Let (A, S, B) be a separation of J where $A = J \cap C$, $S = \hat{R}_i$, and $B = J \setminus V(H)$. By definition of $\hat{\mathcal{B}}$ and $\hat{L}$, there is a set of x edges with at least one endpoint in B such that deleting these from B makes the graph induced on $B \cup S$ bipartite with a bipartition extending $(\hat{R}_{i_1}, \hat{R}_{i_2})$. Since $\hat{\mathcal{B}}$ by definition extends this bipartition, the fact that $\hat{\mathsf{sol}}_{\hat{\mathcal{P}}}$ is a solution to $(\mathcal{I}'_b, \hat{\mathcal{P}})$ implies that there is a set of at most $\ell - x$ edges with both endpoints in $V(H)$ which makes the components of $H - \hat{\mathsf{sol}}_{\hat{\mathcal{P}}}$ intersecting $\hat{R}_i$ bipartite with a bipartition extending $(\hat{R}_{i_1}, \hat{R}_{i_2})$. Applying Observation 9.9, we conclude that J contains an edge-oct of size at most ℓ, a contradiction. This completes the argument for the second case and we now move on to the final case.

Case 3: $J \cap T' \neq \emptyset$ and $J \cap T \neq \emptyset$. First of all, observe that there is a $\hat{R}_i \in \hat{\mathcal{R}}$ and an $R_j \in \mathcal{R}$ such that $J \cap Q = \hat{R}_i$ and $R_j \supseteq (\hat{R}_i \cap T)$. Furthermore, the bipartition (R_{j_1}, R_{j_2}) induced by $\mathcal{B}$ is compatible with the bipartition $(\hat{R}_{i_1}, \hat{R}_{i_2})$ induced by $\hat{\mathcal{B}}$.

We partition the set $J' = J \setminus (T \cup T')$ into the following 3 sets.

- J_1= the vertices in $J' \cap V(H)$.
- J_2= the vertices in $J' \cap \mathrm{H}^G_{\mathcal{P}}(R_j)$.
- J_3= the vertices in $J' \setminus V(H)$.

Since J_3 is disjoint from $Y_{\mathcal{P}}$ and $Y_{\mathcal{P}}$ contains the vertices in $X_{\mathcal{P}} \setminus V(H)$, we conclude that J_3 is disjoint from $X_{\mathcal{P}}$. Since the vertices in J_2 are undeletable, the set $X_{\mathcal{P}}$ is by definition disjoint from J_2. Therefore, vertices in J_3 and J_2 appear in the same connected component of $G_{\mathcal{P}} - X_{\mathcal{P}}$ and there is a set E' of $y \leq \ell$ edges in the graph $K = G_{\mathcal{P}}[J_2 \cup J_3 \cup \hat{R}_i \cup (R_j \setminus V(H))]$ whose deletion from K makes the resulting graph bipartite with a bipartition respecting $(\hat{R}_{i_1}, \hat{R}_{i_2})$ and (R_{j_1}, R_{j_2}). Since there are no edges between J_2 and J_3, each edge in E' with an endpoint in $J_2 \cup J_3$ has an endpoint in either J_2 or J_3 but never both. Let y_1 and y_2 be such that E' contains y_1 edges with an endpoint in J_2 and y_2 edges with an endpoint in J_3. By definition of $\hat{\mathcal{P}}$, we have that $\hat{L}$ contains the pair $(\hat{R}_i, y)$. Since $\hat{\mathsf{sol}}_{\hat{\mathcal{P}}}$ is a solution for $(\mathcal{I}'_b, \hat{\mathcal{P}})$, we have that there is a set of at most $\ell - y$ edges with both endpoints in H such that deleting these makes the connected components of $H - \hat{\mathsf{sol}}_{\hat{\mathcal{P}}}$ intersecting $\hat{R}_i$, bipartite with a bipartition extending $(\hat{R}_{i_1}, \hat{R}_{i_2})$. Combining these edges along with the y_1 edges guaranteed by Lemma 9.8 and the aforementioned y_2 edges with an endpoint in J_2, we obtain a set of at most ℓ edges which makes the component J in $G_{\mathcal{P}} - Y_{\mathcal{P}}$, bipartite. This completes the argument for the final case and hence we conclude that $Y_{\mathcal{P}}$ is indeed a solution for $(\mathcal{I}_b, \mathcal{P})$.

Since $Y_{\mathcal{P}}$ by definition has the property that $Y_{\mathcal{P}} \cap C \subseteq Z$, we have completed the proof of first part of the lemma. For the second part, we observe that if there exists v such that $v \in \mathsf{sol}_{\mathcal{P}'}$ for all $\mathcal{P}' \in \mathbb{P}(\mathcal{I}'_b)$, then $v \in Y_{\mathcal{P}}$ and hence v is part of a minimum sized solution for $(\mathcal{I}_b, \mathcal{P})$ for all $\mathcal{P} \in \mathbb{P}(\mathcal{I}_b)$. This completes the proof of the lemma. $\square$

Before we state our next lemma, we need to recall the notion of *parity-torso* (see for example [LR12]).

Definition 9.7. *Let G be a graph and $S \subseteq V(G)$. We denote by $PT(G,S)$ the graph obtained from $G - S$ as follows. For every pair of vertices u, v in $V(G) \setminus S$, if there is an odd u-v path whose internal vertices all lie in S then we add an edge (u,v) and if there is an even u-v path whose internal vertices all lie in S then we add a* subdivided *edge (path of length 2) between u and v.*

The utility of this operation follows from the observation below.

Observation 9.11. *Let G be a graph, S and X be disjoint vertex sets. Let A be an edge set with both endpoints of every edge disjoint from S. Suppose that for every connected component K in $G[S]$, for every $v \in N(K)$, the graph $G[K \cup \{v\}]$ is bipartite. Then, for any $v \in V(G) \setminus S$, there is an odd cycle in the connected component of v in $PT(G,S) - (X \cup A)$ if and only if there is an odd cycle in the connected component of v in $G - (X \cup A)$.*

Proof. Since no edge in A is incident on S, it follows that $PT(G,S) - (X \cup A) = PT(G - (X \cup A), S)$. Therefore, it suffices to prove that for any $Z \subseteq V(G)$ with the property that every connected component of $G[Z]$ along with any single neighbor induces a bipartite graph, for any $v \notin Z$, there is an odd cycle in the connected component of v in G if and only if there is an odd cycle in the connected component of v in $PT(G,Z)$. It is clear from the definition of the parity torso that u and v are in the same connected component of $PT(G,Z)$ if and only if they are in the same connected component of G.

In the forward direction, let C be an odd cycle in G in the connected component containing v. By the premise of the lemma, this cycle contains at least 2 vertices of $V(G) \setminus Z$. Let P be any subpath of C between vertices $u_1, u_2 \in V(G) \setminus Z$ with all internal vertices in Z. Then, the definition of $PT(G,Z)$ implies either an edge (if $|P_1|$ is odd) or a subdivided edge (if $|P_1|$ is even) between u_1 and u_2 in $PT(G,Z)$. Thus, we can replace every such subpath with the corresponding edge or subdivided edge to obtain an odd cycle in $PT(G,Z)$. Note that the property regarding the bipartiteness of any connected component of $G[Z]$ along with a single neighbor is critical since otherwise the statement clearly fails.

We now argue the converse direction. Let C be an odd cycle in $PT(G,Z)$ in the same component as v. If C is also in G then we are done. Therefore, there is at least one edge in C that is not in G. Let $P_1, \ldots, P_r$ be all the subpaths of C which correspond to either edges or subdivided edges added in the construction of $PT(G, Z)$. We can replace each P_i with the corresponding path in G to obtain a closed odd walk in G in the same connected component as v, implying an odd cycle in this component. $\square$

We are now ready to state our next crucial lemma which essentially gives a way to *get rid of* vertices which we know will never be required in our solution.

Lemma 9.12. *Let $\mathcal{I}_b = (G, k, \ell, U, T)$ be an instance of* B-PBD. *Let $T' \subseteq V(G) \setminus U$ and let C be a connected component of $G - T'$ such that $N(C) = T'$. Let $H = G[C \cup T']$ and let $Q = T' \cup (C \cap T)$ and suppose that $|Q| \leq 2k$. Let $\mathcal{I}'_b$ denote the* B-PBD *instance $(H, k, \ell, (U \cap V(H), Q)$. Suppose that for every $v \in V(H)$, there is a $\mathcal{P} \in \mathbb{P}(\mathcal{I}'_b)$ such that $v \notin \mathsf{sol}_{\mathcal{P}}$. Let Z denote the set $\bigcup_{\mathcal{P} \in \mathbb{P}(\mathcal{I}'_b)} \mathsf{sol}_{\mathcal{P}}$. Then,*

- *$|Z| = 2^{\mathcal{O}(k \log(k+\ell))}$ and*
- *there are functions τ, α and an algorithm that, given $\mathcal{I}_b$ and Z, runs in time $(\tau(k,\ell) n^{\mathcal{O}(1)})$ and computes a set $W \subseteq V(H)$ such that $W \supseteq Q$ and has size at most $\alpha(k,\ell)$ such that the instance $\mathcal{I}_b$ is equivalent to the instance $\mathcal{I}_b^1 = (G', k, \ell, U, T)$ where the graph G' is defined as $PT(G, V(H) \setminus W)$. Here, $\alpha(k,\ell)$ and $\tau(k,\ell)$ are both $2^{\mathcal{O}(k \log(k+\ell))}$.*

Proof. For the first statement, we observe the following.

$$\begin{aligned}
|\mathbb{P}(\mathcal{I}_b)| &\leq (\ell+1)^{|T|+1}(1 + 2(|T|+1))^{|T|} \\
&\leq (\ell+1)^{2k+1}(1 + 2(2k+1))^{2k} \\
&= 2^{\mathcal{O}(k \log(k+\ell))}
\end{aligned}$$

This is true because $\mathcal{R}$ has at most $|T|+1$ equivalence classes, $\mathcal{B}$ has at most 2 equivalence classes, each $v \in T$ can either go to X_b or choose an equivalence class in $\mathcal{R}$ and $\mathcal{B}$, and L has $(\ell+1)^{|T|+1}$ possible values.

We now prove the second statement. For each $\mathcal{P} \in \mathbb{P}(\mathcal{I}'_b)$ such that $\mathsf{sol}_{\mathcal{P}}$ exists, we define a set $W_{\mathcal{P}}$ as follows. Let $\mathcal{J}$ denote the set of connected components of $H_{\mathcal{P}} - \mathsf{sol}_{\mathcal{P}}$. For each connected component $J \in \mathcal{J}$, we let $E^J_{\mathcal{P}}$ denote a set of at most ℓ edges whose deletion makes J bipartite (such a set exists since $\mathsf{sol}_{\mathcal{P}}$ is an ℓ-pseudobipartite deletion set for $H_{\mathcal{P}}$). We let $V^J_{\mathcal{P}}$ denote those endpoints of these edges which are contained in $V(H)$. We define $W_{\mathcal{P}}$ to be the the set $\bigcup_{J \in \mathcal{J}: J \cap (Z \cup Q) \neq \emptyset} V^J_{\mathcal{P}}$. Finally, we define W to be the union of the sets Q, Z and $\bigcup_{\mathcal{P} \in (\mathbb{P}(\mathcal{I}'_b))} W_{\mathcal{P}}$.

Before we go ahead and define the instance $\mathcal{I}^1_b$, we prove the following claim which is necessary to invoke Observation 9.11 at various points in our argument.

Claim 1. *Let K be a connected component in $G[V(H) \setminus W]$. Then, for any $v \in N(K)$, the graph $G[K \cup \{v\}]$ is bipartite.*

Proof. Suppose that this is not the case and there is a $v \in N(K)$ such that the graph $G[K \cup \{v\}]$ and hence the graph $H[K \cup \{v\}]$ is non-bipartite. By the premise of the lemma, we know that for some $\mathcal{P} \in \mathbb{P}(\mathcal{I}'_b)$, $v \notin \mathsf{sol}_{\mathcal{P}}$. This implies that $K \cup \{v\}$ is part of a connected component of $H_{\mathcal{P}} - \mathsf{sol}_{\mathcal{P}}$. We now consider 2 cases: $v \in Z$ or $v \notin Z$. In the first case, observe that by definition, we have added *both* endpoints of some edge-oct of the graph $G[K \cup \{v\}]$ to $W_{\mathcal{P}}$, a contradiction to the graph $G[K \cup \{v\}]$ being non-bipartite. In the second case, the reason v is added to W is because there is a $\mathcal{P}' \in \mathbb{P}(\mathcal{I}'_b)$ such that v is one of the endpoints of the edge set of a minimum edge-oct of the component containing v in $H_{\mathcal{P}'} - \mathsf{sol}_{\mathcal{P}'}$. Again, since $K \cup \{v\}$ is part of such a component and W contains *both* endpoints of an edge-oct of this graph, it cannot be the case that $G[K \cup \{v\}]$ is non-bipartite. This completes the proof of the claim. $\square$

We now prove that the instance $\mathcal{I}_b$ is equivalent to the instance $\mathcal{I}^1_b = (G', k, \ell, U, T)$ where $G' = PT(G, V(H) \setminus W)$. That is, for every $\mathcal{P} \in \mathbb{P}(\mathcal{I}_b)$, there is a solution for $(\mathcal{I}_b, \mathcal{P})$ if and only if there is a solution for $(\mathcal{I}^1_b, \mathcal{P})$.

In the forward direction, suppose that for some $\mathcal{P} \in \mathbb{P}(\mathcal{I}_b)$, there is a solution $X_{\mathcal{P}}$ for $(\mathcal{I}_b, \mathcal{P})$. By Lemma 10.14, we may assume without loss of generality that $X_{\mathcal{P}} \cap V(H) \subseteq Z$ and furthermore, there is a $\hat{\mathcal{P}} \in \mathbb{P}(\mathcal{I}'_b)$ such that $X_{\mathcal{P}} \cap V(H) = \hat{\mathsf{sol}}_{\hat{\mathcal{P}}}$. Hence, $X_{\mathcal{P}} \subseteq V(G')$. We claim that $X_{\mathcal{P}}$ itself is a solution for $(I^1_b, \mathcal{P})$. If this were not the case then there is a component J in $G'_{\mathcal{P}} - X_{\mathcal{P}}$ which has no edge-oct of size at most ℓ. Observe that if J

is disjoint from W by Observation 9.11, then it is also part of a connected component in $G_{\mathcal{P}} - X_{\mathcal{P}}$ which has no edge-oct of size at most ℓ, a contradiction. Therefore, J intersects W. Now, if J is disjoint from Q, then Observation 9.11 implies that it is part of a connected component of $H_{\hat{\mathcal{P}}} - \hat{\mathsf{sol}}_{\hat{\mathcal{P}}}$ (see proof of Lemma 10.14 for definition of $\hat{\mathcal{P}}$), with no edge-oct of size at most ℓ a contradiction to $\hat{\mathsf{sol}}_{\hat{\mathcal{P}}}$ being a solution for $(\mathcal{I}'_b, \hat{\mathcal{P}})$. Thus we conclude that J must intersect Q and is contained in a connected component of $H_{\hat{\mathcal{P}}} - \hat{\mathsf{sol}}_{\hat{\mathcal{P}}}$. Let $J' \supseteq J$ be the connected component of $H_{\hat{\mathcal{P}}} - \hat{\mathsf{sol}}_{\hat{\mathcal{P}}}$ which contains the vertices of J. By definition, W contains both endpoints of *some* edge-oct of size at most ℓ for the component J'. By Observation 9.11, it follows that J also has an edge-oct of size at most ℓ, a contradiction. This completes the argument in the forward direction.

The converse direction follows directly from Observation 9.11. Observe that the size of W is bounded by 2ℓ times the size of Z. Furthermore, given Z, we need to perform at most $|Z| \cdot |\mathbb{P}(\mathcal{I}'_b)|$-many edge-oct computations with each edge-oct being bounded by ℓ. For this, we can use the algorithm in [GGH+06] and hence the size and running time bounds follow, completing the proof of the lemma. $\square$

Now we describe the recursive step of the algorithm.

Step 3. Assume we are given a B-PBD instance $\mathcal{I}_b = (G, k, \ell, U, T)$ and let $q := \alpha(k,\ell) + \binom{\alpha(k,\ell)}{2} + 1$. Invoke first the algorithm of Lemma 10.3 in a search for (q,k)-good node separation (with $V^\infty = U$). If it returns a good node separation (Z, V_1, V_2), let $j \in \{1,2\}$ be such that $|V_j \cap T| \leq k$ and denote $T' = N(V_j) \subseteq Z$, $C = V_j$. Otherwise, if it returns that no such good node separation exists in G, invoke the algorithm of Lemma 10.4 in a search for (q,k)-flower separation w.r.t. T_b (with $V^\infty = U$ again). If it returns that no such flower separation exists in G, pass the instance $\mathcal{I}_b$ to the next step (high connectivity phase). Otherwise, if it returns a flower separation $(Z, (V_i)_{i=1}^r)$, denote $C = \bigcup_{i=1}^r V_i$ and $T' = N(C) \subseteq Z$. Let $H = G[C \cup T']$. In the case we have obtained T' and C (either from Lemma 10.3 or Lemma 10.4), invoke the algorithm recursively for the B-PBD instance $\mathcal{I}'_b$ defined as in the statement of Lemma 10.14 for sets T' and set C, obtaining an output $\mathsf{sol}_{\mathcal{P}}$ for each $\mathcal{P} \in \mathbb{P}(\mathcal{I}'_b)$. If there exists $v \in V(H)$ such that $v \in \mathsf{sol}_{\mathcal{P}}$ for every $\mathcal{P} \in \mathbb{P}(\mathcal{I}'_b)$, we return v as the special vertex. Otherwise, compute the set $Z = \bigcup_{\mathcal{P} \in \mathbb{P}(\mathcal{I}'_b)} \mathsf{sol}_{\mathcal{P}}$. Use the algorithm of Lemma 9.12 on $\mathcal{I}'_b$ and Z to compute the set W. Generate the graph $G' = PT(G, V(H) \setminus W)$. Let $\mathcal{I}^1_b = (G', k, \ell, (U \cap V(H)), \mathcal{Q})$, where $\mathcal{Q} = T' \cup (C \cap T)$.

Restart this step on instance $\mathcal{I}^1_b$. If it returns a special vertex v, then return v as a special vertex for the instance $\mathcal{I}_b$. Otherwise, obtain a family of solutions $(\mathsf{sol}'_{\mathcal{P}})_{\mathcal{P} \in \mathbb{P}}$ and return this family as output to the instance $\mathcal{I}_b$.

We first show that application of Lemma 10.14 and Lemma 9.12 is justified in Step 8. By definitions of good node separations and good flower separations and by choice of C and T', we have that $V(H) \cap T \leq k$ and that H is connected. Also, the recursive calls are applied to strictly smaller graphs, as in case of finding a good node separation, V_2 is deleted when we make the recursive call, and in the case of finding a flower separation, we have that $Z \cup \bigcup_{i=1}^{r} V_i$ is a strict subset of $V(G)$. Now we state the following lemma which argues the correctness of Step 8.

Lemma 9.13. *Assume that we are given a* B-PBD *instance $\mathcal{I}_b = (G, k, \ell, U, T)$ on which Step 8 is applied, and let $\mathcal{I}'_b$ be an instance after Step 8 is applied. Then any correct output (either a special vertex or a set of solutions) to the instance $\mathcal{I}'_b$ is a correct output to the instance $\mathcal{I}_b$ as well. Moreover, if Step 8 outputs $\perp$ for all $\mathcal{P} \in \mathbb{P}(\mathcal{I}'_b)$, then this is a correct output to $\mathcal{I}_b$.*

The proof of the lemma follows from Lemma 10.14 and Lemma 9.12. Now we do a running time analysis for Step 8. Since $q = \mathcal{O}(2^{\mathcal{O}(k \log(k+\ell))})$, finding a good (q,k)-node separation or flower separation takes time $\mathcal{O}(2^{\mathcal{O}(\mathsf{min}(q,k)\log(q+k))} n^3 \log n) = \mathcal{O}(2^{\mathcal{O}(k^2 \log(k+\ell))} n^3 \log n)$. Let $|V(H)| = n'$, and hence by definitions of good node separation and flower separation we have that $q+1 \leq n' \leq n-q-1$. The first recursive call is applied to an instance on n' vertices. While taking the torso operation, we have that $|W| = \alpha(k,\ell) = 2^{\mathcal{O}(k \log(k+\ell))}$ and finding the set W takes $\tau(k,\ell) n^{\mathcal{O}(1)} = 2^{\mathcal{O}(k \log(k+\ell))} n^{\mathcal{O}(1)}$ time.

Let $H' = G - V(H)$. We know that T' separates H' from rest of the graph and $T' \subseteq W$. So, for any $u, v \in V(G)$ such that $u \in V(H')$, we do not have any path from u to v having its internal vertices entirely in $V(H) \setminus W$. So if a new vertex z is added because of an even length path from u to v, we have that $u, v \in V(H)$. As $G - (V(H) \setminus W)$ has at most

$n - n' + \alpha(k,\ell)$ vertices and none of the vertices in H' contribute to the torso operation, we have that $|V(G')| \leq n - n' + \alpha(k,\ell) + \binom{\alpha(k,\ell)}{2} < |V(G)|$. The base case for the recursive calls is the high connectivity phase, which we will argue takes time $2^{\mathcal{O}((k+\ell)^3(\log k+\ell))}n^{\mathcal{O}(1)}$. Hence, we get the following recurrence for running time of Step 8.

$$T(n) \leq \mathsf{max}_{q+1\leq n' \leq n-q-1}\big(\mathcal{O}(2^{\mathcal{O}(k^2\log(k+\ell))}n^3\log n) + T(n') + 2^{\mathcal{O}(k\log(k+\ell))}n^{\mathcal{O}(1)} + \\ T(\mathsf{min}(n-1, n-n'+q)) + 2^{\mathcal{O}((k+\ell)^3\log(k+\ell))}n^{\mathcal{O}(1)}\big)$$

Solving the recurrence gives $T(n) = 2^{\mathcal{O}((k+\ell)^3\log(k+\ell))}n^{\mathcal{O}(1)}$ in the worst case, which is the running time for Step 8. We remark that we never actually introduce any new undeletable vertices in the graph in this step.

9.4.3 High Connectivity phase

In this section we describe the high connectivity phase of the algorithm. Assume we have a B-PBD instance $\mathcal{I}_b = (G,k,\ell,U,T)$ where Step 8 is not applicable. Let us fix $\mathcal{P} = (X_T,\mathcal{R},\mathcal{B},L) \in \mathbb{P}(\mathcal{I}_b)$ and let $U' := U \cup ((T\setminus X_T)\cup R_{new})$, where $R_{new} = V(G_{\mathcal{P}})\setminus V(G)$. We iterate through all possible values of $\mathcal{P}$ and try to find a minimum solution to $(G_{\mathcal{P}},k,\ell,U')$. Since $|\mathbb{P}(\mathcal{I}_b)| = 2^{\mathcal{O}(k\log(k+\ell))}$, this results in a factor of $2^{\mathcal{O}(k\log(k+\ell))}$ in the running time. Thus, from now onwards we focus on one such $\mathcal{P}$. Furthermore, here we only solve the instances where $|U'| \leq \mathcal{O}(k\ell)$, which is sufficient for our purpose as we will argue later. We first prove the following lemma.

Lemma 9.14. *Let $\mathcal{I}_b = (G,k,\ell,U,T)$ be an instance of* B-PBD *where Step 8 is not applicable. Let $U' := U \cup ((T\setminus X_T)\cup R_{new})$, where $R_{new} = V(G_{\mathcal{P}})\setminus V(G)$. Then for every $\mathcal{P} \in \mathbb{P}(\mathcal{I}_b)$, the graph $G_{\mathcal{P}}$ satisfies the following.*

- *for any $Z \subseteq (V(G_{\mathcal{P}})\setminus U')$ of size at most k, the graph $G_{\mathcal{P}} - Z$ contains at most $f(k,\ell) = (2q+2)(2^k-1)+2k+1$ connected components containing a vertex of $V(G)\setminus U'$, out of which at most one has more than $h(k,\ell)$ vertices not in U' where $h(k,\ell) := q(4k(2k+3)(\ell+1)+1)$, and*

- *$G_{\mathcal{P}}$ has an oct of size at most $g(k,\ell) := 2k + k\ell^2 + 1 + |U'|$.*

Proof. For any $Z \subseteq (V(G_{\mathcal{P}}) \setminus U')$, since $V(G_{\mathcal{P}}) \setminus V(G) \subseteq U'$, we have that $Z \subseteq V(G) \setminus U$. Since Step 8 is not applicable on (G, k, ℓ, U, T), from Lemma 10.5 we know that G does not have a (q,k)-good node separation or (q,k)-flower separation. This gives us that the graph $G - Z$ contains at most $(2q+2)(2^k - 1) + |T| + 1$ connected components containing a vertex of $V(G) \setminus U$, out of which at most one has more than $h(k,\ell)$ vertices not in U.

Let $C_0, C_1, \ldots, C_s$ be the connected components of $G - Z$. Since by construction of the graph $G_{\mathcal{P}}$, all the vertices in $V(G_{\mathcal{P}}) \setminus V(G)$ are either adjacent to one of the vertices in T or adjacent to some other vertex in $V(G_{\mathcal{P}}) \setminus V(G)$ which is adjacent to a vertex in T and $Z \cap (V(G_{\mathcal{P}}) \setminus V(G)) = \emptyset$, we have that the number of connected components of $G_{\mathcal{P}} - Z$ is not larger than that of $G - Z$. This shows that $G_{\mathcal{P}} - Z$ contains at most $(2q+2)(2^k - 1) + |T| + 1$ connected components containing a vertex of $V(G_{\mathcal{P}}) \setminus U'$. Now, let $C'_0, C'_1, \ldots, C'_t$ be the connected components of $G_{\mathcal{P}} - Z$. Since G is a subgraph of $G_{\mathcal{P}}$, for all $i \in \{0, 1, \ldots, s\}$, there exists $j \in \{0, 1, \ldots, t\}$ such that $C_i \subseteq C'_j$. Without loss of generality, let us assume that if $G - Z$ contains a connected component with more than q vertices not in U, then it is contained inside C'_0. Now we argue that for all $i \in [t]$, $|C'_i \setminus U'| \leq h(k,\ell)$. Since $R_{new} \subseteq U'$, only the vertices in $V(G)$ can contribute to the size of $C_i \setminus U'$. We know by construction of $G_{\mathcal{P}}$ that $|E(G_{\mathcal{P}}) \setminus E(G)| \leq 4k(2k+3)(\ell+1)$, so any connected component in $G_{\mathcal{P}} - Z$ can be formed by combining at most $4k(2k+3)(\ell+1)+1$ connected components of $G - Z$. Also, we know that for all $i \in [t]$, all the connected components of $G - Z$ contained in C'_i have at most q vertices from the set $V(G_{\mathcal{P}}) \setminus U$, and hence we have $|C'_i \setminus U'| \leq q(4k(2k+3)(\ell+1)+1)$. This concludes the proof of the first part the lemma.

The second part of the lemma follows from the fact that $V(G_{\mathcal{P}}) \setminus V(G) \subseteq U'$ and that because of Lemma 9.6, G has an oct of size at most $2k + k\ell^2 + 1$. $\square$

So, now we can generate an instance of Pseudobipartite Deletion where an oct is also given and we have a bound on number of vertices from $V(G) \setminus U$ in all connected components after deleting a solution except one. We recall the formal problem definition.

OCT-PBD **Parameter(s):** k,ℓ

Input: An instance (G, k, ℓ, U) of Pseudobipartite Deletion along with an oct O of G of size at most $g(k, \ell)$ such that for any $Z \subseteq (V(G) \setminus U)$ of size at most k, in the graph $G - Z$, at most one connected component containing a vertex of $V(G) \setminus U$ has more than $h(k, \ell)$ vertices not in U.

Output: A minimum sized ℓ-pseudobipartite deletion set X of G of size at most k such that $X \cap U = \emptyset$. Output $\perp$ if such a set does not exist.

Step 4. Find an oct O of $G_{\mathcal{P}}$ of size $g(k, \ell)$ using algorithm in [LNR+14]. Return an instance (G, k, ℓ, U, O) of OCT-PBD.

The correctness of Step 4 is immediate from Lemma 9.14. Now, we branch into the possibilities for the intersection of the set O with the solution of the Pseudobipartite Deletion instance $(G_{\mathcal{P}}, k, \ell, U')$. Recall that we get the following problem.

OCT-PBD(I) **Parameter(s):** k,ℓ

Input: An instance (G, k, ℓ, U, O) of OCT-PBD.

Output: A minimum sized ℓ-pseudobipartite deletion set X of G of size at most k such that $X \cap (O \cup U) = \emptyset$. Output $\perp$ if such a set does not exist.

Now, to solve an OCT-PBD instance we do the following. For each $X_O \subseteq O \setminus U$ of size at most k, we solve the OCT-PBD(I) instance $(G - X_O, k, \ell, U, O \setminus X_O)$. Let $P(X_O)$ be the solution returned for the choice X_O. Then we output the set of the form $X_O \cup P(X_O)$ which has the minimum size such that $|X_O \cup P(X_O)| \leq k$ and output $\perp$ if such a set does not exist. We need to perform this sanity check as in the recursive call we are still looking for minimum solutions of size at most k and the call might return a minimum solution for the graph with more than $k - |O_X|$ vertices.

This way, we branch into $\sum_{0 \leq i \leq k} \binom{|O|}{i}$ cases and it results into a factor of $\sum_{0 \leq i \leq k} \binom{|O|}{i}$ in the running time. The correctness of this step can be seen by looking at the intersection of the solution to OCT-PBD instance (G, k, ℓ, U, O) with the set O. Observe that $O \setminus O_X$ still remains to be an oct for the graph $G - O_X$ and the graph $G - O_X$, being a subgraph of G, still satisfies the high connectivity requirements in the definition of OCT-PBD.

Now we need to solve OCT-PBD(I). To that end, we guess which vertices of the oct are going to be in the same connected component after deleting the solution. We first prove the following lemma.

Lemma 9.15. *Let (G, k, ℓ, U, O) be an instance of* OCT-PBD(I)*. Let $\mathbb{S}$ be the set of all equivalence relations on O. Then, there exists $\S \in \mathbb{S}$ and $X' \subseteq V(G) \setminus (U \cup O)$ such that X' is also a minimum sized ℓ-pseudobipartite deletion set of G and for any $u, v \in O$, u and v belong to the same connected component of $G - X'$ if and only if $(u, v) \in \S$.*

Proof. Let $X \subseteq V(G) \setminus (U \cup O)$ be a minimum sized ℓ-pseudobipartite deletion set of G. We define an equivalence relation $\S'$ on O as following. For any $u, v \in O$, we say that $(u, v) \in \S'$ if and only if u and v belong to the same connected component of $G - X$. Clearly, $\S' \in \mathbb{S}$ and there exists an $X' = X \subseteq V(G) \setminus (U \cup O)$ which is a minimum sized ℓ-pseudobipartite deletion set of G and partitions the vertices of O in the desired way. □

So, to solve an OCT-PBD(I) instance (G, k, ℓ, U, O), for each $\S \in \mathbb{S}$, we try to solve the instance with the guarantee that there exists a solution after deleting which the vertices of the oct are in the same connected component if and only if they are in the same equivalence class of $\S$. More formally, for each $\S \in \mathbb{S}$, we get an instance of the following problem.

OCT-PBD(II) **Parameter(s):** k, ℓ

Input: An instance (G, k, ℓ, U, O) of OCT-PBD(I) and an equivalence relation $\S$ on O with the guarantee that there exists a minimum sized ℓ-pseudobipartite deletion set $X \subseteq V(G) \setminus (U \cup O)$ of G such that for all $u, v \in O$, u and v belong to the same connected component of $G - X$ if and only if $(u, v) \in \S$.

Output: A minimum sized ℓ-pseudobipartite deletion set X of G of size at most k such that $X \cap (O \cup U) = \emptyset$. Output $\perp$ if such a set does not exist.

Now, to solve an OCT-PBD(I) instance (G, k, ℓ, U, O), for each $\S \in \mathbb{S}$, we solve an instance $(G, k, \ell, U, O, \S)$ of OCT-PBD(II) and return the solution to the instance which outputs solution of minimum size. The correctness of this step follows from Lemma 9.15. Since the number of equivalence relations on O is bounded by $g(k, \ell)^{g(k,\ell)}$, we get a factor of $g(k, \ell)^{g(k,\ell)}$ in the running time.

Now to solve OCT-PBD(II). We return No if $k < 0$. We first look at the case when the equivalence relation § has more than one equivalence class. In this case, we know that there exists a solution X after deleting which, at least one of the equivalence classes of § is in a connected component containing at most $h(k, \ell)$ vertices not from U.

We proceed by guessing this equivalence class S_i in §. Then we arbitrarily pick a vertex v in S_i and look at a connected subgraph H of G containing v which has $h(k, \ell) + 1$ vertices not from U. We know that at least one of the vertices in $V(H)$ has to be part of the solution X, because after deleting X, the connected component containing v has at most $h(k, \ell)$ vertices not from U. Then we pick a vertex of $V(H)$ and branch on it. Since each branching call decreases solution size we are looking for by at least one, the depth of the recursion tree is bounded by k.

If such a subgraph H does not exist, we have that the connected component C containing v has at most $h(k, \ell)$ vertices not in U, and then we solve the problem on $G[C]$ using brute force, which takes time $h(k, \ell)^k n^{\mathcal{O}(1)}$.

Now we deal with the case when § has only one equivalence class. That is, we know that there exists a solution $X \subseteq V(G) \setminus (U \cup O)$ such that $G - X$ is ℓ-pseudobipartite and for all $(u, v) \in O$, u and v belong to the same connected component of $G - X$. In other words, there exists a solution X of minimum size such that all the vertices of O lie in the same connected component C of $G - X$. To solve this problem, we first prove the following.

Lemma 9.16. *Let $(G, k, \ell, U, O, \S)$ be an* OCT-PBD(II) *instance such that for all $u, v \in O$, $(u, v) \in \S$. Then there exists a bipartition $(O_1 \uplus O_2)$ and $X' \subseteq V(G) \setminus (U \cup O)$ such that X' is a minimum sized ℓ-pseudobipartite deletion set of G, all vertices of O belong to the same connected component C of $G - X'$ and there exists an edge-oct F of $G[C]$ of size at most ℓ and a bipartition $(C_1 \uplus C_2)$ of C such that $G[C_1] - F$ and $G[C_2] - F = \emptyset$ are independent sets and $O_1 \subseteq C_1$ and $O_2 \subseteq C_2$.*

Proof. Since for all $u, v \in O$, $(u, v) \in \S$, we know by the definition of OCT-PBD(II) that there exists a minimum sized ℓ-pseudobipartite deletion set $X^* \subseteq V(G) \setminus (U \cup O)$ of G such that all the vertices of O belong to the same connected component of $G - X$. Let F^* be the edge-oct of size at most ℓ of C. Then we know that $G[C] - F^*$ is bipartite.

Hence, there exists a bipartition $(C_1^* \uplus C_2^*)$ of C such that $G[C_1^*] - F = G[C_2^*] - F = \emptyset$. Let $(O_1^* \uplus O_2^*)$ be the bipartition induced on O by $(C_1^* \uplus C_2^*)$. Then $(O_1 \uplus O_2) = (O_1^* \uplus O_2^*)$, $X' = X^*$, $F = F^*$ and $(C_1 \uplus C_2) = (C_1^* \uplus C_2^*)$ satisfy the conditions of the lemma. $\square$

This lemma helps us reduce an instance of OCT-PBD(II) into $2^{|O|}$ instances of following problem, defined in the overview section.

OCT-PBD(III) **Parameter(s):** k,ℓ

Input: An instance $(G, k, \ell, U, O, \S)$ of OCT-PBD(II) such that for all $u, v \in O$, $(u, v) \in \S$ and a bipartition $(O_1 \uplus O_2)$ of O with the guarantee that there exists a minimum sized ℓ-pseudobipartite deletion set $X \subseteq V(G) \setminus (U \cup O)$ of G such that all vertices of O belong to the same connected component C of $G - X$ and there exists an edge-oct F of $G[C]$ of size at most ℓ and a bipartition $(C_1 \uplus C_2)$ of C such that $G[C_1] - F$ and $G[C_2] - F$ are independent sets and $O_1 \subseteq C_1$ and $O_2 \subseteq C_2$.

Output: A minimum sized ℓ-pseudobipartite deletion set X of G of size at most k such that $X \cap (O \cup U) = \emptyset$. Output $\perp$ if such a set does not exist.

Now, to solve an instance $(G, k, \ell, U, O, \S)$ of OCT-PBD(II), for each bipartition $(O_1 \uplus O_2)$ of O, we solve the OCT-PBD(III) instance $(G, k, \ell, U, O, \S, (O_1, O_2))$ and return the solution to the instance which outputs solution of the minimum size. The correctness of this step follows from Lemma 9.16. Since there are $2^{|O|}$ bipartitions of O, this step branches into $2^{|O|}$ instances of OCT-PBD(III) for each instance of OCT-PBD(II).

Now, to solve an instance $I := (G, k, \ell, U, O, \S, (O_1 \uplus O_2))$ of OCT-PBD(III), we define a graph G_I as follows. Let $(A \uplus B)$ be an arbitrary bipartition of $V(G) \setminus O$ such that $E((V(G) - O)[A]) = E((V(G) - O)[B]) = \emptyset$ and let $\ell' = |E(G[O_1]) \cup E(G[O_2])|$. To get the graph G_I, we first make two copies of vertex set O, namely O_A and O_B. Now, we define the vertex set of $V(G_I)$ to be the set $(V(G) \setminus O) \cup O_A \cup O_B$. We represent copy of a vertex $v \in O$ in O_A by v_A and the copy in O_B by v_B. Then we make every vertex $v_A \in O_A$ adjacent to a vertex $w \in A$ if and only if v and w are adjacent in G. Similarly, we make every vertex in $v_B \in O_B$ adjacent to a vertex $w \in B$ if and only if v and w are adjacent in G. We copy the edges of $G - O$ as it is to $G_I - (O_A \cup O_B)$. Let $O_A^i = \{v_A | \ v \in O_i\}$

and $O^i_B = \{v_B | \ v \in O_i\}$ for $i \in \{1,2\}$. Finally, let $S^* := O^1_A \cup O^2_B$, $T^* := O^2_A \cup O^1_B$ and $O^* := S^* \cup T^*$. We make the induced subgraphs $G_I[S^*]$ and $G_I[T^*]$ into cliques. This finishes the construction of the graph G_I. We define the equivalence relation $\mathcal{R}_I$ on the set O^* as follows. For $u, v \in O^*$, $(u,v) \in \mathcal{R}_I$ if and only if $u, v \in S^*$ or $u, v \in T^*$.

Observe that there exists a natural one to one correspondence between $E(G) \setminus E(G[O])$ and $E(G_I) \setminus E(G[S^* \cup T^*])$. That is, we have made exactly one copy of each of the edges which do no have both endpoints in O. Let us first recall the definition of MMCU* from Section 9.3 and state the theorem about the running time of the algorithm to solve MMCU* (which we prove independently in the next chapter).

MMCU* **Parameter(s):** k,ℓ

Input: A graph G, integers k and ℓ, $T \subseteq V(G)$, an equivalence relation $\mathcal{R}$ on T having at most two equivalence classes, and set of undeletable vertices $U \subseteq V(G)$.

Output: Output a minimal solution $\mathcal{X} = (X, F)$ to MMCU instance $(G, T, \mathcal{R}, k, \ell)$ such that $X \cap U = \emptyset$ and $\perp$ if no such solution exists.

Theorem 9.8 (Threorem 10.10 in Chapter 10)**.** MMCU* *can be solved in time* $(|U|+2)^{|U|}\mathcal{O}(2^{\mathcal{O}((k+\ell)^3 \log(k+\ell))}n^4 \log n)$.

Now we prove the following two lemmas.

Lemma 9.17. *Let X be a solution of* OCT-PBD(III) *instance* $I := (G, k, \ell, U, O, \S, (O_1 \uplus O_2))$ *of size* $k' \leq k$. *Then the algorithm for* MMCU* *instance* $(G_I, k', \ell - \ell', O^*, \mathcal{R}_I, U)$ *outputs a solution.*

Proof. We look at the graph $G' := G - (X, F)$. Since O is an oct of G and F is an edge-oct of connected component containing O, we have that G' is bipartite as well. We look at the bipartition $(C_1 \uplus C_2)$ of connected component C of $G - X$ as guaranteed by the definition of OCT-PBD(III). Let $V' = V(G')$. Since all the other connected components are already bipartite, there exists a bipartition $(V_1 \uplus V_2)$ of V' such that $E(G'[V_1]) = E(G'[V_2]) = \emptyset$ and $O_i \subseteq V_i$ for $i \in \{1,2\}$. So, in G', any path from a vertex to O_i to a vertex in O_i has even length for $i \in \{1,2\}$ and any path from O_1 to O_2 has odd length.

Let $F' := F \cap E(G - O)$. As F must contain $E(G[O_1]) \cup E(G[O_2])$, $|F'| \leq \ell - \ell'$ and all the edges in F' have at least one end point in $V(G) \setminus O$. It suffices to show that (X, F') is a solution to MMCU* instance $(G_I, k', \ell - \ell', O^*, \mathcal{R}_I, U)$. Since F' does not contain any edges from the cliques $G_I[S^*]$ and $G_I[T^*]$ we have that any $u, v \in S^*$ remains in the same connected component of $(G_I - X) - F'$ and any $u, v \in T^*$ also remain in the same connected component of $G_I - (X, F')$. Now we need to show that for any $u \in S^*$ and $v \in T^*$, there does not exist a path from u to v in $G_I - (X, F')$.

For the sake of contradiction, let us assume that such a path does exist. We look at such a path such that all the internal vertices are disjoint from $S^* \cup T^*$. Since these paths only contain edges such that they have at least one endpoint in $V(G_I) \setminus O^*$, we can talk about their corresponding paths in G as well. Let x and y be the first and last vertices for such a path. Since $S^* = O_A^1 \cup O_B^2$ and $T^* = O_B^1 \cup O_A^2$, there are four possibilities for such a path. Let us look at the case when $x \in O_A^1$ and $y \in O_B^1$. As $G_I - O^*$ is bipartite, it is easy to see that this path has odd length. But as we have taken $F' := F \cap E(G - O)$, the path corresponding to this in G is also present in G', and is a path from a vertex of O_1 to a vertex of O_1 having odd length, which is a contradiction. Similarly, we can argue for the cases when $x \in O_A^1$ and $y \in O_A^2$, $x \in O_B^2$ and $y \in O_B^1$ and $x \in O_B^2$ and $y \in O_A^2$ and arrive at a contradiction. This proves that (X, F') is indeed a solution for MMCU* instance $(G_I, k', \ell - \ell', U, O^*, \mathcal{R}_I)$ and concludes the proof of the lemma. $\square$

Our final lemma helps us in binding all the things together.

Lemma 9.18. *Let $I := (G, k, \ell, U, O, \S, (O_1 \uplus O_2))$ be a* OCT-PBD(III) *instance and $k' \leq k$ be the minimum number for which the algorithm for* MMCU* *outputs a solution (X, F) for the instance $(G_I, k', \ell - \ell', U, O^*, \mathcal{R}_I)$, then X is a correct solution to I as well.*

Proof. Let $I := (G, k, \ell, U, O, \S, (O_1 \uplus O_2))$ be a OCT-PBD(III) instance and let X^* be a minimum sized ℓ-pseudobipartite deletion set of G such that $|X^*| = k'$. Now we need to show that if running the algorithm for MMCU* instance $(G_I, k', \ell - \ell', U, O^*, \mathcal{R}_I)$ outputs (X, F) then X is a an ℓ-pseudobipartite deletion set of G.

In fact, we argue that $G - X$ has an edge-oct of size at most ℓ. It is easy to see that if the solution is minimal, then the set F is disjoint from $E(G[S^* \cup T^*])$, and hence we will be talking about the set F in G as well as in G'. Let $G' = G - X$, then we claim that G' has an edge-oct of size at most ℓ. To show that, it suffices to prove that $F' := F \cup G[O_1] \cup G[O_2]$ is an edge-oct of size at most ℓ of G. That is, we need to show that the graph $(G - X) - F'$ does not have an odd cycle.

First we look at a path from a vertex in O_1 to a vertex in O_1 in $(G - X) - F'$ such that all the internal vertices are in O. Since $G[O_1] \cup G[O_2] \subseteq F'$, we have that this path has even length. Similarly we can show that in $(G - X) - F'$, any path completely contained in O from a vertex in O_2 to a vertex in O_2 has even length and any path completely contained in O from a vertex in O_1 to a vertex in O_2 has odd length. Then we look at any path from any vertex in O_1 to any vertex in O_1 in $(G - X) - F'$ such that all its internal vertices are disjoint from O and look at its corresponding path in G_I. Since (X, F) kills all paths between O_A^1 and O_B^1 and $F \subseteq F'$, this has to be a path either from a vertex in O_A^1 to a vertex in O_A^1 or from a vertex in O_B^1 to a vertex in O_B^1. Since $G_I - O^*$ is bipartite, it is easy to see that all these paths are of even length. Similarly, we can argue that all paths from a vertex in O_2 to a vertex in O_2 such that all internal vertices are not in O are of even length. Now we look at all paths from O_1 to O_2 in $(G - X) - F'$ such that all internal vertices are not from O and look at it's corresponding path in G_I. Since (X, F) kills all paths from O_A^1 to O_A^2 and paths from O_B^1 to O_B^2, the only paths left from a vertex in O_1 to a vertex in O_2 are either from O_A^1 to O_B^2 or from O_B^1 to O_A^2. Again, as $G_I - O^*$ is bipartite and $F \subseteq F'$, it is easy to see that all these paths have odd length.

Let C be a cycle in $G - (X, F')$. If $V(C) \cap O = \emptyset$, then C is a cycle of even length. Let $v_1, v_2, \ldots, v_t$ be the vertices of $C \cap O$ in their order of appearance along C. Let $v_0 = v_t$, then we have that $|E(C)| = \sum_{i=0}^{t-1} d_C(v_i, v_{i+i})$. But then $|E(C)|$ must be even since the number of indices i such that $v_i \in O_1$ and $v_{i+1} \in O_2$ is equal to the number of indices j such that $v_j \in O_2$ and $v_{j+1} O_2$. Hence, any cycle in $(G - X) - F'$ is of even length, and X is an ℓ-pseudobipartite deletion set of G. $\square$

Now we are ready to give the final step of the algorithm. To solve an OCT-PBD(III) instance $I := (G, k, l, U, O, \S, (O_1 \uplus O_2))$, for each $k' \in \{0, 1, \ldots, k\}$, we solve the MMCU*

instance $(G_I, k', \ell - \ell', U, O^*, \mathcal{R}_I)$, where G_I, ℓ', O^* and $\mathcal{R}_I$ are as described above. Let $k^* \in \{0, 1, \ldots, k\}$ be the minimum such number for which the algorithm for MMCU* instance $(G_I, k^*, \ell - \ell', U, O^*, \mathcal{R}_I)$ returns a solution. Let this solution be (X^*, F^*). Return X^*. The correctness of this step follows from Lemma 9.17 and Lemma 9.18.

Now we do the running time analysis of the high connectivity phase assuming the size of the undeletable vertices is bounded by $\mathcal{O}(k\ell)$. First we find an oct of size $g(k,\ell)$ to get an instance of OCT-PBD, which takes time $2^{\mathcal{O}(k^2\ell^4)} n^{\mathcal{O}(1)}$ using the algorithm in [LNR+14] since $g(k,\ell) = \mathcal{O}(k\ell^2)$. Then we branch into solving at most $2^{g(k,\ell)} = 2^{\mathcal{O}(k\ell^2)}$ instances of OCT-PBD(I). For solving each instance of OCT-PBD(I), we branch into $g(k,\ell)^{g(k,\ell)}$ instances of OCT-BPD(II). Since $g(k,\ell) = \mathcal{O}(k\ell^2)$, we have that $g(k,\ell)^{g(k,\ell)} = 2^{\mathcal{O}(k\ell^2 \log(k\ell))}$. Hence, the total running time of high connectivity phase for an instance (G, k, ℓ, U) is $n^{\mathcal{O}(1)} + 2^{\mathcal{O}(k\ell^2 \log(k\ell))} + \beta(k,\ell)$, where $\beta(k,\ell)$ is the time to solve an OCT-PBT(II) instance.

Now for analyzing running time for solving an instance $\mathcal{I} = (G, k, \ell, U, O, \S)$ of OCT-PBD(II), we first look at the case when $\S$ has only one equivalence class. In this case, we branch into at most $2^{|O|} = 2^{\mathcal{O}(k\ell^2(\log k + \log \ell))}$ instances of OCT-PBD(III). After that, we make the graph $G_{\mathcal{I}}$ and solve the MMCU* instance at most k times. It is easy to see that the graph $G_{\mathcal{I}}$ can be constructed in polynomial time. Also, from Theorem 9.8, the running time for MMCU* instance is $(|U| + 2)^{|U|} \mathcal{O}(2^{\mathcal{O}((k+\ell)^3 \log(k+\ell))} n^4 \log n)$. Hence, the total running time for OCT-PBD(II) is bounded by $2^{\mathcal{O}((k+\ell)^3 \log(k+\ell))} n^{\mathcal{O}(1)}$ in the case when $\S$ has only one equivalence class. Now, when the equivalence relation $\S$ has more than one equivalence class, then we either solve the problem on the connected component containing v by brute force or we branch into $g(k,\ell)$ cases, by guessing the equivalence class which belongs to a component which has bounded number of vertices from $V(G) \setminus U$. Picking a vertex v and finding a connected subgraph containing v having $h(k,\ell) + 1$ undeletable vertices can be done in polynomial time. Then we either decrease the number of connected components of the graph in $h(k,\ell)^k n^{\mathcal{O}(1)}$ time or we branch into $h(k,\ell) + 1$ cases, where in each of them the solution size we are looking for drops by one. The base case is when the solution size we are looking for becomes negative and we can say No or when the instance has only one equivalence class, in which case we know how to solve it in

$(|U|+2)^{|U|}2^{\mathcal{O}((k+\ell)^3\log(k+\ell))}n^4\log n$ time. Hence, the total running time for OCT-PBD(II) is bounded by $2^{\mathcal{O}((k+\ell)^3\log(k+\ell))}n^{\mathcal{O}(1)}$ as we have that the number of undeletable vertices is bounded by $\mathcal{O}(k\ell)$.

This gives that the total running time for high connectivity phase is $2^{\mathcal{O}((k+\ell)^3\log(k+\ell))}n^{\mathcal{O}(1)}$ as we solve at most $2^{\mathcal{O}(k\log(k+\ell))}$ instances of Pseudobipartite Deletion. This finishes the description of the algorithm. Now we are ready to prove the final theorem.

Proof. **(of Theorem 9.1)** For solving an instance (G^*,k,ℓ) of Strong Bipartite Deletion we first reduce it to an instance (G^*,k,ℓ,U^*) of Pseudobipartite Deletion by putting $U^*=\emptyset$. Then we use the preprocessing rules described in Section 9.4.1 to get $2^{\mathcal{O}(k)}$ many instances of Pseudobipartite Deletion, such that the graph in each instance has a bounded sized oct. As argued earlier, this takes time $2^{\mathcal{O}(k\ell^2)}n^{\mathcal{O}(1)}$.

Then we apply Step 2 and solve the problem on connected components of the graph. This results in at most k many instances and can be applied in $1.977^{\ell}n^{\mathcal{O}(1)}$ time, Now, let (G,k,ℓ,U) be an instance of Pseudobipartite Deletion after application of Step 2. Then we know that G is connected, so we generate an instance $(G,k,\ell,U,\emptyset)$ of B-PBD and solve it. If we find a special vertex, then by Observation 9.7, we know that this vertex is part of some minimum solution for the Pseudobipartite Deletion instance (G,k,ℓ,U). Hence we can safely delete this vertex from the graph and apply the algorithm on $(G-\{v\},k-1,\ell,U)$ starting from Step 2. Observe that the oct bound on the graph still holds. Since each time we find a special vertex, the budget decreases by one, we do this process at most k many times. We have already given the description and the correctness of the algorithm for B-PBD. We observe that the set of undeletable vertices is initially empty, and the only time we actually add undeletable vertices to a graph is while solving the high connectivity phase, in which case, we add at most $2k(4\ell+1)=\mathcal{O}(k\ell)$ vertices to the undeletable set. Hence, the running time argued for the high connectivity phase is valid, which takes $2^{\mathcal{O}((k+\ell)^3\log(k+\ell))}n^{\mathcal{O}(1)}$ time. This finishes the proof of the theorem. □

Chapter 10

Parameterized Algorithm for Mixed Cut

10.1 Introduction

Given a graph, a typical *cut problem* asks for finding a set of vertices or edges such that their removal from the graph makes the graph satisfy some separation property. The most fundamental version of the cut problems is Minimum Cut, where given a graph and two vertices, called *terminals*, we are asked to find the minimum sized subset of vertices (or edges) of the graph such that deleting them separates the terminals. The Minimum Cut problem is known to be polynomial time solvable for both edge and vertex versions and both in undirected and directed graphs. The core of the polynomial time solvability of the Minimum Cut problem is one of the classical min-max results in graph theory – the Menger's theorem. The classical Menger's theorem states that in any undirected (or directed) graph G, given a pair of vertices s and t, the maximum number of vertex (edge) disjoint paths is equal to the minimum number of vertices (edges) needed to disconnect from s and t.

While Minimum Cut is polynomial time solvable; even a slight generlization becomes NP-hard. Two of the most studied generalizations of Minimum Cut problem which are NP-hard are Multiway Cut and Multicut. In the Multiway Cut problem, we are given a set of terminals, and we are asked to delete minimum number of vertices (or edges) to separate the terminals from each other. This problem is known to be NP-hard even when the number of terminals is at least three. In the MULTICUT problem, given pairs of terminals, we are asked to delete minimum number of vertices (or edges) so that it separates all the given terminal pairs. The MULTICUT problem is known to be NP-hard when the number of pairs of terminals is at least three. The mixed version of the problem, which is the central topic of this paper, namely MIXED CUT is also NP-hard. In this problem we are given an undirected graph G, vertices s and t, positive integers k and ℓ and the objective is to test whether there exist a k sized vertex set $S \subseteq V(G)$ and an ℓ sized edge set $F \subseteq E(G)$ such that deletion of S and F from G disconnects from s and t. In this chapter we study MIXED CUT, in fact a stronger generlization of it in the realm of parameterized complexity. In this chapter we mainly study the following problem.

MIXED MULTIWAY CUT-UNCUT (MMCU) **Parameter(s):** k, ℓ

Input: A multigraph G, a set of terminals $T \subseteq V(G)$, an equivalence relation $\mathcal{R}$ on the set T and integers k and ℓ.

Question: Does there exist $X \subseteq (V(G) \setminus T)$ and $F \subseteq E(G)$ such that $|X| \leq k$, $|F| \leq \ell$ and for all $u, v \in T$, u and v belong to the same connected component of $G - (X, F)$ if and only if $(u, v) \in \mathcal{R}$?

We start by giving a brief overview of related work and then give our results and methods.

Related Works. Cut problems were looked at under the realm of Parameterized Complexity by Marx [Mar06] for the first time, who showed that MULTIWAY CUT is FPT when parameterized by the solution size and MULTICUT is FPT when parameterized by the solution size plus the number of terminals. Subsequently, a lot of work has been done on cut problems in the field of parameterized complexity [BDT11, CLL09, CHM13, KT11, KPPW12, MOR13, MR14]. Recently, Chitnis et al. [CCH+12] introduced the technique of *randomized contractions* and used that to solve the UNIQUE LABEL COVER problem. They also show that the same techniques can be applied to solve a generalization of MULTIWAY CUT problem, namely MULTIWAY CUT-UNCUT, where an equivalence relation $\mathcal{R}$ is also supplied along with the set of terminals and we are to delete minimum number of vertices (or edges) such that the terminals lie in the same connected of the resulting graph if and only if they lie in the same equivalence class of $\mathcal{R}$. It is easy to see that MMCU not only generalized Mixed Cut and Mixed Multiway Cut, but also both edge and vertex versions of Multiway Cut and Multiway Cut-Uncut problems. Mixed Cut is studied and mentioned in the books [BW13, Fra12] and is also a useful subroutine in parameterized graph editing problems. Cao and Marx [CM14] studied this problem during their study on Chordal Editing problem and gave an algorithm with running time $2^{\mathcal{O}(k+\ell)}n^{\mathcal{O}(1)}$ on chordal graphs. Algorithms for cut-problems can be applied to several problems, which at first do not look like cut problems. Examples include well studied problems such as Feedback Vertex Set [CLL+08] and Odd Cycle Transversal [RSV04]. We have already seen in Chapter 9 that algorithm for a special version of the MMCU problem with undeletable vertices is used as a subroutine in the algorithm for Pseudobipartite Deletion. Hence, it is natural and timely to obtain a parameterized algorithms for MMCU.

Our Results and Methods. Even though the vertex and edge versions of Minimum Cut problem are polynomial time solvable, we show that allowing deletion of both, the vertices and the edges, makes the Mixed Cut problem NP-hard. To show that, we use a simple reduction from the Bipartite Partial Vertex Cover problem which was recently shown to be NP-hard [AS14, JV12]. Then we show that MMCU is FPT. In particular we prove the following theorem.

Theorem 10.1. MMCU *is* FPT *with an algorithm running in time* $2^{(k+\ell)^{\mathcal{O}(1)}} \cdot n^{\mathcal{O}(1)}$.

There are two ways to approach our problem – one is via treewidth reduction technique of Marx et al. [MOR13] and the second is via the method of recursive understanding introduced by Chitnis et al. [CCH+12]. Applying both these methods have its own obstacles that we need to tackle. However, the method of treewidth reduction technique would lead to an algorithm for MMCU that has double exponential dependence on $k + \ell$ and thus we do not pursue this method. As we did in the last chapter, we use recursive understanding introduced by Chitnis et al. [CCH+12] to solve the problem. The main observation is that if there is a small vertex separation which divides the graph into big parts, then we can recursively reduce the size of one of the big parts. Otherwise, the graph is highly connected, and the structure of the graph can be exploited to obtain a solution for MMCU. We follow the framework given in [CCH+12] and design our algorithm. In particular we utilise the recursive understanding technique to first find a small separator in the graph which separates the graph into two parts, each of sufficiently large size and then recursively solve a 'border' version of the same problem on one of the two sides. The border version of the problem is a generalization which also incorporates a special bounded set of vertices, called terminals. During the course of our algorithm, we will attempt to solve the border problem on various subgraphs of the input graph. The objective in the border problem is to find a bounded set of vertices containing within a particular subgraph such that any vertex in this subgraph *not* in the computed set is not required in *any* solution for the given instance irrespective of the vertices chosen outside this subgraph. The algorithm in [CCH+12] returns the minimum solutions in the recursive steps. Since we allow both edge and vertex deletion, there is no clear ordering on the solutions, and hence we need to look for solutions of all possible sizes while making the recursive call.

This leaves us with the base case of the recursion, that is when we are unable to find a separator of the required kind. This is called high connectivity phase and this is the place where one needs problem specific algorithm in the framework given in [CCH+12]. Since the solution we are looking for contains both edges and vertices, we need some additional work, as the good node separation framework gives bound only for vertices that can be part of the solution. Once we have done that, the frameworks lends itself for our use, and we can use a separating set family to get to the solution.

In the end, we solve a special version of MMCU with undeletable vertices, namely MMCU*, which completes the picture for our algorithm for Pseudoforest Deletion.

10.2 Preliminaries

In this section, we first give the notations and definitions which are used in the chapter. Then we state some basic properties of mixed-cuts and some known results which will be used later in the chapter.

We define the Mixed Cut and Mixed Multiway Cut-Uncut problems as follows.

Mixed Cut	**Parameter(s):** k, ℓ
Input: A multigraph G, vertices $s, t \in V(G)$, integers k and ℓ.	
Question: Does there exist $X \subseteq V(G)$ and $F \subseteq E(G)$ such that $\lvert X\rvert \leq k$, $\lvert F\rvert \leq \ell$ and s and t are in different connected components of $G - (X, F)$?	

Mixed Multiway Cut-Uncut (MMCU)	**Parameter(s):** k, ℓ
Input: A multigraph G, a set of terminals $T \subseteq V(G)$, an equivalence relation $\mathcal{R}$ on the set T and integers k and ℓ.	
Question: Does there exist $X \subseteq (V(G) \setminus T)$ and $F \subseteq E(G)$ such that $\lvert X\rvert \leq k$, $\lvert F\rvert \leq \ell$ and for all $u, v \in T$, u and v belong to the same connected component of $G - (X, F)$ if and only if $(u, v) \in \mathcal{R}$?	

We say that a tuple $\mathcal{X} = (X, F)$, where $X \subseteq V(G) \setminus T$ and $F \subseteq E(G)$, is a solution to a MMCU instance $\mathcal{I} = (G, T, \mathcal{R}, k, \ell)$ if $\lvert X\rvert \leq k$, $\lvert F\rvert \leq \ell$ and for all $u, v \in T$, u and v belong to the same connected component of $G - (X, F)$ if and only if $(u, v) \in \mathcal{R}$. We define a partial order on the solutions of the instance $\mathcal{I}$. For two solutions $\mathcal{X} = (X, F)$ and $\mathcal{X}' = (X', F')$ of a MMCU instance $\mathcal{I}$, we say that $\mathcal{X}' \leq \mathcal{X}$ if $X' \subseteq X$ and $F' \subseteq F$. We say that a solution $\mathcal{X}$ to an MMCU instance $\mathcal{I}$ is *minimal* if there does not exist another solution $\mathcal{X}'$ to $\mathcal{I}$ such that $\mathcal{X}' \neq \mathcal{X}$ and $\mathcal{X}' \leq \mathcal{X}$. For a solution $\mathcal{X} = (X, F)$ of an MMCU instance $\mathcal{I} = (G, T, \mathcal{R}, k, \ell)$ and $v \subseteq V(G)$, we say that $\mathcal{X}$ *affects* v if either $v \in X$ or there exists $u \in V(G)$ such that $uv \in F$.

Observation 10.1. *If $\mathcal{X} = (X, F)$ is a minimal solution to a* MMCU *instance $\mathcal{I} = (G, T, \mathcal{R}, k, \ell)$, then none of the edges in F are incident to X.*

Now we recall the definitions of good node separations and flower separations from [CCH+12] given in Chapter 9. Then we state the lemmas that state the running time to find such separations and the properties of the graph if such separations do not exist.

Lemma 10.2 ([CCH+12])**.** *Given a set U of size n together with integers $0 \leq a, b \leq n$, one can in $\mathcal{O}(2^{\mathcal{O}(\mathsf{min}(a,b)\log(a+b))} n \log n)$ time construct a family $\mathcal{F}$ of at most $\mathcal{O}(2^{\mathcal{O}(\mathsf{min}(a,b)\log(a+b))} \log n)$*

subsets of U, such that the following holds: for any sets $A, B \subseteq U$, $A \cap B = \emptyset$, $|A| \leq a$, $|B| \leq b$, there exists a set $S \in \mathcal{F}$ with $A \subseteq S$ and $B \cap S = \emptyset$.

Definition 10.2 ([CCH+12])**.** *Let G be a connected graph and $V^\infty \subseteq V(G)$ a set of undeletable vertices. A triple (Z, V_1, V_2) of subsets of $V(G)$ is called a (q,k)-good node separation, if $|Z| \leq k$, $Z \cap V^\infty = \emptyset$, V_1 and V_2 are vertex sets of two different connected components of $G - Z$ and $|V1 \setminus V^\infty|, |V_2 \setminus V^\infty| > q$.*

Definition 10.3 ([CCH+12])**.** *Let G be a connected graph, $V^\infty \subseteq V(G)$ a set of undeletable vertices, and $T_b \subseteq V(G)$ a set of border terminals in G. A pair $(Z, (V_i)_{i=1}^r)$ is called a (q,k)-flower separation in G (with regard to border terminals T_b), if the following holds:*

- *$1 \leq |Z| \leq k$ and $Z \cap V^\infty = \emptyset$; the set Z is the core of the flower separation $(Z, (V_i)_{i=1}^r)$;*
- *V_i are vertex sets of pairwise different connected components of $G - Z$, each set V_i is a petal of the flower separation $(Z, (V_i)_{i=1}^r)$;*
- *$V(G) \setminus (Z \cup \bigcup_{i=1}^r V_i)$, called a stalk, contains more than q vertices of $V \setminus V^\infty$;*
- *for each petal V_i we have $V_i \cap T_b = \emptyset$, $|Vi \setminus V^\infty| \leq q$ and $N_G(V_i) = Z$;*
- *$|(\bigcup_{i=1}^r V_i) \setminus V^\infty| > q$.*

Lemma 10.3 ([CCH+12])**.** *Given a connected graph G with undeletable vertices $V^\infty \subseteq V(G)$ and integers q and k, one may find in $\mathcal{O}(2^{\mathcal{O}(\min(q,k)\log(q+k))} n^3 \log n)$ time a (q,k)-good node separation of G, or correctly conclude that no such separation exists.*

Lemma 10.4 ([CCH+12])**.** *Given a connected graph G with undeletable vertices $V^\infty \subseteq V(G)$ and border terminals $T_b \subseteq V(G)$ and integers q and k, one may find in $\mathcal{O}(2^{\mathcal{O}(\min(q,k)\log(q+k))} n^3 \log n)$ time a (q,k)-flower separation in G w.r.t. T_b, or correctly conclude that no such flower separation exists.*

Lemma 10.5 ([CCH+12])**.** *If a connected graph G with undeletable vertices $V^\infty \subseteq V(G)$ and border terminals $T_b \subseteq V(G)$ does not contain a (q,k)-good node separation or a (q,k)-flower separation w.r.t. T_b then, for any $Z \subseteq V(G) \setminus V^\infty$ of size at most k, the graph $G - Z$ contains at most $(2q+2)(2^k - 1) + |T_b| + 1$ connected components containing a vertex of $V(G) \setminus V^\infty$, out of which at most one has more than q vertices not in V^∞.*

10.3 NP-completeness of Mixed Cut

We prove that Mixed Cut in NP-complete by giving a reduction from the Bipartite Partial Vertex Cover problem which is defines as follows.

Bipartite Partial Vertex Cover (BPVC)	
Input:	A bipartite graph $G = (X \uplus Y, E)$, integers p and q
Output:	Does there exist $S \subseteq V(G)$ such that $\|S\| \leq p$ and at least q edges in E are incident on X?

Theorem 10.4 ([AS14, JV12]). BPVC *is NP-complete.*

For an instance of BPVC, we assume that the given bipartite graph does not have any isolated vertices, as a reduction rule can be applied in polynomial time which takes care of isolated vertices and produces an equivalent instance. Given an instance (G, p, q) of BPVC where $G = (X \uplus Y, E)$ is a bipartite graph, we get an instance (G', s, t, k, ℓ) of Mixed Cut as follows. To get the graph G', we introduce two new vertices s and t and add all edges from s to X and t to Y. More formally, $G' = (V', E')$ where $V' = V(G) \cup \{s, t\}$ and $E' = E \cup \{sx \mid x \in X\} \cup \{ty \mid y \in Y\}$. Then we put $k = p$ and $\ell = m - q$, where $m = |E|$. It is easy to see that (G, p, q) is a Yes instance of BPVC if and only if (G', s, t, k, ℓ) is a Yes instance of Mixed Cut, and hence we get the following theorem.

Theorem 10.5. Mixed Cut *is NP-complete even on bipartite graphs.*

10.4 An overview of the algorithm

In this section we present an overview of the algorithm for MMCU. We use the idea of Recursive Understanding introduced by Chitnis et al. [CCH+12]. We describe the recursive phase of the algorithm in Section 10.5.3 and the high connectivity phase of the algorithm in Section 10.5.4.

We first perform a few operations on the graph which let us assume that the graph is connected and that the terminals in each of the connected components are divided into bounded number of equivalence classes. For that, we first observe in Lemma 10.6, that if a vertex has paths to terminals in more than $k + \ell + 1$ equivalence classes, then it has to be part of a solution. Then we describe in Lemma 10.7 how to find such a vertex, and then in Lemma 10.8 that if no such vertex exists, then the number of equivalence classes of terminals in a connected component is bounded. In steps 5 and 6, we define the corresponding operations. This part is a direct adaptation of equivalence class reduction techniques from [CCH+12]. Then in Step 7, we show how to take care of the case when the graph is disconnected. Basically, we ask for minimal solutions of all sizes for all the connected components, and combine them in all possible ways to see if a solution exists to original problem.

Now we need to solve the problem on a connected graph where the number of equivalence classes of terminals is bounded. For that, we define a border problem, B-MMCU, where we are additionally provided with a set of border terminals. We can reduce an instance of MMCU to an instance of B-MMCU by just putting this additional set of terminals as the empty set.

Border-Mixed Multiway Cut-Uncut(B-MMCU) **Parameter(s):** k, ℓ

Input: An MMCU instance $\mathcal{I} = (G, T, \mathcal{R}, k, \ell)$ with G being connected and a set $T_b \subseteq V(G) \setminus T$ such that $|T_b| \leq 2(k + \ell)$; denote $\mathcal{I}_b = (G, T, \mathcal{R}, k, \ell, T_b)$.

Output: For each $\mathcal{P} = (X_b, E_b, \mathcal{R}_b, k', \ell') \in \mathbb{P}(\mathcal{I}_b)$, output a $\mathsf{sol}_{\mathcal{P}} = \mathcal{X}_{\mathcal{P}}$ being a minimal solution to $(\mathcal{I}_b, \mathcal{P})$, or $\mathsf{sol}_{\mathcal{P}} = \perp$ if no solution exists.

Here, the set $\mathbb{P}(\mathcal{I}_b)$ denotes the set of 'interactions' of the border terminals of instance $\mathcal{I}_b$ with a solution and the objective is to find a solution that corresponds for each possible interaction. Since we are looking for a solution for each possible interaction, we would want the set of interactions, $\mathbb{P}(\mathcal{I}_b)$, to have size bounded by some function of k and ℓ. For that, we make use of the guarantee that the number of equivalence classes is bounded.

As is the case in algorithms based on the recursive understanding approach, to solve the border problem, we proceed to check whether a good separator Z exists in the graph. We use the notions of good node separations and good flower separations defined by Chitnis et al.[CCH+12] and look for good $(q, k + \ell)$ node separation or $(q, k + \ell)$ flower separation, where q is sufficiently large to guarantee sufficient reduction of vertices (but is still bounded by a function of k and ℓ). The definitions are given in the preliminaries section. The running times required to compute such separations (if they exist) are given by Lemmas 10.3 and 10.4. If we succeed in finding such a separation, then we see that we have divided the graph into two large parts using a small number of vertices. The definitions of node separations and flower separation help us argue that one of the parts (which contains at most half of the border terminals) is connected. We call the smaller graph G^*.

Now we update the set of terminals to include the separator, and solve the border problem $\mathcal{I}_b^*$ recursively on the smaller graph G^*. That is, for every behavior $\mathcal{P} \in \mathbb{P}(\mathcal{I}_b^*)$ of the new border terminals, we get an optimum solution. Let $U(\mathcal{I}_b^*)$ denote the set such that each vertex in it is either in a solution returned by the recursive call to certain behavior of the border terminals, or has an edge incident on it which is in the solution returned by the the recursive call. Then in Lemma 10.14, we show that a solution exists which intersects with the smaller graph only in the vertices of $U(\mathcal{I}_b^*)$. Then we use the bypassing operation defined in Definition 10.6 to get rid of all vertices in smaller graph except the ones in $U(\mathcal{I}_b^*)$. Lemma 10.9 justifies the bypassing operation.

Now, to argue that we have reduced the number of vertices by a significant amount, we also need to argue that the number of terminals in the small graph are bounded. For that, we observe in Lemmas 10.10 and 10.11 that we can apply some operations when two terminals are adjacent. Note that the identifying operation in Lemma 10.11 can introduce new parallel edges, and is the reason that we need multigraphs. After application of these operations, we know that any terminal has neighborhood only in the in the non-terminal vertices. In this case, in Lemma 10.12 we argue that not many of them can have the same neighborhood, and hence we can bound the number of terminals as well and get the significant reduction in number of vertices we were aiming for. In Lemma 10.15, we argue that applying the above mentioned steps actually preserves the solutions.

Since the size of the set $\mathbb{P}(\mathcal{I}_b)$ can be bounded by some function of k and ℓ, and for each of them we need to keep some bounded number of vertices, we can guarantee that after the application of the recursive step (Step 8 in the algorithm) the number of vertices in the graph decreases by a sufficiently large number.

We then describe the algorithm for the problem in the case when there is no good-separation to recurse upon. This is done in Section 10.5.4. Here, we need to solve the B-MMCU instance $\mathcal{I}_b = (G, T, \mathcal{R}, k, \ell, T_b)$ in the case that Step 8 is not applicable on the graph G. In this case, we try to find a minimal solution to each $\mathcal{P} \in \mathbb{P}(\mathcal{I}_b)$. Since size of $\mathbb{P}(\mathcal{I}_b)$ is bounded by a function of k and ℓ, this results in the same factor in the running time of the high-connectivity step.

Now we would want to exploit the high connectivity structure of the graph to solve the problem. We know because of Lemma 10.5, that after deleting the solution in the graph G, the number of connected components having a non-terminal is bounded, and also the number of non-terminals in all such connected components except one. The idea is to try to use the sets provided by Lemma 10.2 to highlight the solution. But for that, we need to bound the number of vertices *and* edges of the connected components (except one). We already know that the number of non-terminals is bounded. Now, to get a bound on number of edges, first we use Lemmas 10.10, 10.11 and 10.12 to get a bound on number of terminals in these connected components. Then we use Lemma 10.13 to finally bound the number of edges in these connected components. Then we prove Lemma 10.16 to argue that there exists a set family of bounded size provided by Lemma 10.2 which highlights the solution.

Finally, in Lemma 10.17, we argue that these set families can be used to get to a solution of the instance $(\mathcal{I}_b, \mathcal{P})$. Essentially, we show that it is enough to look at the neighborhood of the *small* connected components of the graph G after deleting the solution. Then we argue the running time of the high connectivity phase, and show that it is bounded by a function of k and ℓ times some polynomial function in n.

10.5 An algorithm for MMCU

In this section, we describe the FPT algorithm for MMCU. In fact, we will give an algorithm, which when provided with an instance $(G, T, \mathcal{R}, k, \ell)$ of MMCU, not only decides whether there exists a solution (X, F) such that $|X| \leq k$ and $|F| \leq \ell$, but also outputs such a solution that is also minimal. To that end, we first describe how we can bound the number of equivalence classes in a connected component of the given graph.

10.5.1 Reduction of Equivalence Classes

We first show that if there exists a vertex which has a large number of vertex disjoint paths to terminals which are in different equivalence classes, then that vertex has to be part of the solution.

Lemma 10.6. *Let $\mathcal{I} = (G, T, \mathcal{R}, k, \ell)$ be a* MMCU *instance and let $v \in V(G) \setminus T$. Assume that there exist $k + \ell + 2$ paths $P_1, P_2, \ldots, P_{k+\ell+2}$ in G, such that:*

- *for each $1 \leq i \leq k + \ell + 2$, the path P_i is a simple path that starts at v and ends at $v_i \in T$;*
- *any two paths P_i and P_j, $i \neq j$, are vertex disjoint except for the vertex v.*
- *for any $i \neq j$, $(v_i, v_j) \notin \mathcal{R}$.*

Then for any solution (X, F) of $\mathcal{I}$ we have $v \in X$.

Proof. For a contradiction assume that $v \notin X$. Since for $i \neq j$, any two paths P_i and P_j are vertex disjoint except for the vertex v, we have that after we delete vertices of X and edges of F there are still two paths remaining among the given collection, say P_a and P_b. But then we get a path from some $v_a \in T$ to some vertex $v_b \in T$ in $G - (X, F)$ by concatenating P_a and P_b such that $(v_a, v_b) \notin \mathcal{R}$. This contradicts the fact that (X, F) is a solution to $\mathcal{I}$. ☐

Lemma 10.7. *Let $\mathcal{I} = (G, T, \mathcal{R}, k, \ell)$ be a* MMCU *instance. For any $v \in V(G)$, we can verify of v satisfies conditions of Lemma 10.6 in $\mathcal{O}((k + \ell)n^2)$ time.*

Proof. To verify whether a vertex $v \in V(G)$ satisfies the conditions of Lemma 10.6 we do the following. Given a graph G we transform it into a directed network D as follows. For every edge $uv \in E(G)$ we have two directed arcs (u, v) and (v, u) in $E(D)$ and for every equivalence class $Z \in \mathcal{R}$ we make a new vertex t_z and give a directed arc from $z \in Z$ to t_z (that is we add (z, t_z)). Finally, we add a super-sink t and give directed arc (t_z, t) for every $Z \in \mathcal{R}$. Furthermore, give every arc unit capacity. Now to check whether v satisfies the conditions of Lemma 10.6 all we need to check whether there is a flow of size $k + \ell + 2$ in D from v to t. This can be done by running $k + \ell + 2$ rounds of augmenting path and thus the running time is upper bounded by $\mathcal{O}((k + \ell + 2) \cdot (|E(D)| + |V(D)|))$. Thus the claimed running time follows. □

Step 5. *For each $v \in V(G)$, if v satisfies the conditions of Lemma 10.6, delete v from the graph and decrease k by one; if k becomes negative by this operation, return* No*. Then restart the algorithm.*

Using the algorithm described in Lemma 10.7, each application of Step 5 takes $\mathcal{O}((k+\ell)n^3)$ time. As Step 5 can not be applied more than k times, all applications of the step take $\mathcal{O}(k(k + \ell)n^3)$ time. Now we show how application of Step 5 gives a bound on number of equivalence classes.

Lemma 10.8. *Let $\mathcal{I} = (G, T, \mathcal{R}, k, \ell)$ be the instance of* MMCU *which we get after the exhaustive application of Step 5. If there exists a connected component in G containing terminals from more than $(k + \ell)(k + \ell + 1)$ equivalence classes of $\mathcal{R}$, then $\mathcal{I}$ is a* No *instance of* MMCU*.*

Proof. For a contradiction assume that $\mathcal{I}$ is a Yes instance of MMCU. We will show that if this is the case then there exists an opportunity to apply Step 5. Let (X, F) be a solution to $\mathcal{I}$ and X' be a set of vertices containing all of X and exactly one vertex from each of the edges in F. Clearly, the size of X' is at most $k + \ell$. Let $C_1, \ldots, C_s$ be the connected components of $G - X'$ that contains at least one terminal vertices. Since G contains terminals from more than $(k + \ell)(k + \ell + 1)$ equivalence classes of $\mathcal{R}$ and each connected component of $G - X'$ contains vertices from at most one equivalence class we have that $s > (k + \ell)(k + \ell + 1)$. Thus by Pigeon Hole Principle there exists a vertex $v \in X'$ that has neighbors in at least $k+\ell+2$ connected components. These connected components together with v imply that we could apply Step 5. This contradicts our assumption that on $\mathcal{I}$ Step 5 is no longer applicable. Thus indeed $\mathcal{I}$ is a No instance of MMCU. $\square$

Step 6. *If there exist $u, v \in T$ such that u and v lie in different connected components of G or there exists a connected component of G with terminals of more than $(k+\ell)(k+\ell+1)$ equivalence classes or $\mathcal{R}$, return* No.

Correctness of Step 6 follows from Lemma 10.8 and it can be applied in $\mathcal{O}(n^2)$ time. In the next step, we take care of disconnected graphs which helps us assume that the input graph is connected.

Step 7. *Let $C_1, C_2, \ldots C_t$ be the vertices corresponding to the connected components of the graph G in the input instance $\mathcal{I} = (G, T, R, k, \ell)$. For each C_i and for every $k' \in \{0, 1, \ldots, k\}$ and $\ell' \in \{0, 1, \ldots, \ell\}$ we pass the instance $\mathcal{I}^i_{k',\ell'} = (G, T \cap C_i, \mathcal{R}|_{T\cap C_i}, k', \ell')$ to the next step. If for every $i \in \{1, \ldots, t\}$ there exists $\mathcal{I}^i_{k'_i,\ell'_i}$ such that the subroutine returns (X_i, F_i) for $\mathcal{I}^i_{k'_i,\ell'_i}$ and $\sum_{i\in\{1,\ldots,t\}} k'_i \leq k$, $\sum_{i\in\{1,\ldots,t\}} \ell'_i \leq \ell$, then return $(\bigcup_{i\in\{1,\ldots,t\}} X_i, \bigcup_{i\in\{1,\ldots,t\}} F_i)$, otherwise return* No.

The correctness of this step is easy to see, as after the application of Step 6, each equivalence class is confined to at most one connected component of the graph. The step can be applied in $\mathcal{O}(n^2)$ time and gives rise to $\mathcal{O}(kln)$ subproblems. After application of Step 7, we can assume that G is connected and that the number of equivalence classes in G are bounded by $(k+\ell)(k+\ell+1)$.

10.5.2 Operations on the Graph

Definition 10.6. *Let $\mathcal{I} = (G, T, \mathcal{R}, k, \ell)$ be a* MMCU *instance and let $v \in V(G) \setminus T$. By bypassing a vertex v we mean the following operation: we delete the vertex v from the graph and, for any $u_1, u_2 \in N_G(v)$, we add an edge (u_1, u_2) if it is not already present in G.*

Definition 10.7. *Let $\mathcal{I} = (G, T, \mathcal{R}, k, \ell)$ be an* MMCU *instance and let $u, v \in T$. By identifying vertices u and v in T, we mean the following operation: we make a new set*

$T' = (T \setminus \{u, v\}) \cup \{x_{uv}\}$ and for each edge of the form $uw \in E(G)$ or $vw \in E(G)$, we add an edge $x_{uv}w$ to $E(G)$. Observe that the operation might add parallel edges.

Lemma 10.9. *Let $\mathcal{I} = (G, T, \mathcal{R}, k, \ell)$ be a* MMCU *instance, let $v \in V(G) \setminus T$ and let $\mathcal{I}' = (G', T, \mathcal{R}, k, \ell)$ be the instance $\mathcal{I}$ with v bypassed. Then:*

- *if $\mathcal{X} = (X, F)$ is a solution to $\mathcal{I}'$, then $\mathcal{X}$ is a solution to $\mathcal{I}$ as well;*
- *if $\mathcal{X} = (X, F)$ is a solution to $\mathcal{I}$ and $v \notin X$ and for all $u \in N(v)$ $vu \notin F$ then $\mathcal{X}$ is a solution to $\mathcal{I}'$ as well.*

Proof. We first prove the first item in the lemma. Suppose $\mathcal{X} = (X, F)$ is a solution to $\mathcal{I}'$ but it is not a solution to $\mathcal{I}$. This implies that either there exists $(x, y) \in \mathcal{R}$ and x and y belong to two different connected components of $G - \mathcal{X}$ or $(x, y) \notin \mathcal{R}$ and they are in the same connected component of $G - \mathcal{X}$. In the first case we know that x and y belong to the same connected component of $G' - \mathcal{X}$ and thus we know that the two connected components of $G - \mathcal{X}$ which contain x and y have neighbors of v. This contradicts that x and y belong to two different connected components of $G - \mathcal{X}$. For the later case if there is a path containing x and y then it must contain v but then in G' we have an edge between the neighbors of v on this path, which in turn would imply a path between x and y in $G' - \mathcal{X}$. One can prove the second statement along the similar lines. □

Lemma 10.10. *Let $\mathcal{I} = (G, T, \mathcal{R}, k, \ell)$ be a* MMCU *instance and let $u, v \in T$ be two different terminals with $(u, v) \notin \mathcal{R}$, such that $uv \in E(G)$, then for any solution $\mathcal{X} = (X, F)$ of $\mathcal{I}$, we have $uv \in F$.*

The proof of the Lemma 10.10 follows from the fact that any solution must delete the edge uv to disconnect u from v. The proof of the next lemma follows by simple observation that u and v have at least $k + \ell + 1$ internally vertex disjoint paths or from the fact that $(u, v) \in \mathcal{R}$ and thus after deleting the solution they must belong to the same connected component and thus every minimal solution does not use the edge $uv \in E(G)$.

Lemma 10.11. *Let $\mathcal{I} = (G, T, \mathcal{R}, k, \ell)$ be a* MMCU *instance and let $u, v \in T$ be two different terminals with $(u, v) \in \mathcal{R}$, such that $uv \in E(G)$ or $|N_G(u) \cap N_G(v)| > k + \ell$. Let*

$\mathcal{I}'$ be instance $\mathcal{I}$ with terminals u and v identified. Then set of minimal solutions of $\mathcal{I}$ and $\mathcal{I}'$ is the same.

Lemma 10.12. *Let $\mathcal{I} = (G, T, \mathcal{R}, k, \ell)$ be a* MMCU *instance and let $U = \{v_1, v_2, \ldots, v_t\} \subseteq T$ be different terminals of the same equivalence class of $\mathcal{R}$, pairwise nonadjacent and such that $N_G(u_1) = N_G(u_2) = \cdots = N_G(u_t) \subseteq V(G) \setminus T$ and $t > \ell + 2$. Let $\mathcal{I}'$ be obtained from $\mathcal{I}$ by deleting all but $\ell + 2$ terminals in U (and all pairs that contain the deleted terminals in $\mathcal{R}$). Then the set of minimal solutions to $\mathcal{I}$ and $\mathcal{I}'$ are equal.*

Proof. Without loss of generality, let the set deleted terminals be $U' := \{u_{\ell+3}, \ldots, u_t\}$ and let $N := N_G(u_1) = N_G(u_2) = \cdots = N_G(u_t)$. Let $u, v \in V(G) \setminus U'$. We claim that for any minimal solution $\mathcal{X} = (X, F)$ of $\mathcal{I}$, u and v are in the same connected component of $G - \mathcal{X}$ if and only if they are in the same connected component of $G' - \mathcal{X}$. The backward direction is trivial as $G' - \mathcal{X}$ is a subgraph of $G - \mathcal{X}$. To show the forward direction, we just need to show that F does not contain any edges incident on U, because then if a path visits a vertex in U', then we can redirect it via one of u_i's which remain in the graph. For $v \in X \cap N$, by Observation 10.1, we have that F does not contain any edges incident on v. For $v \in N \setminus X$, there are $u_a, u_b \in U \setminus U'$ which are adjacent to v in $G - \mathcal{X}$. This gives us that v lies in the same connected component of $G - \mathcal{X}$ as U, and hence a minimal solution does not contain any of the edges between x and U. This shows that $\mathcal{X}$ is a solution for $\mathcal{I}'$ as well.

For the converse, let $\mathcal{X}$ be a minimal solution to $\mathcal{I}'$. We just need to show that if $u, v \in T$ such that $v \in T \setminus U'$ and $u \in U'$, they lie in same connected component of $G - \mathcal{X}$ if and only if $(u, v) \in \mathcal{R}$. Since $l \geq 0$, we have that $|U \setminus U'| \geq 2$. Now it is easy to see that $N \subsetneq X$ because otherwise the vertices of $U \setminus U'$ will not be in the same connected component of $G' - \mathcal{X}$. Hence, there exists $x \in N \setminus X$. Since $|N_{G'}(x) \cap U'| = \ell + 2$, x is adjacent to at least 2 vertices of U' in $G' - \mathcal{X}$. Without loss of generality, let $u_1, u_2 \in U \setminus U'$ be these two vertices. Now let $v \in T \setminus U'$ and $u \in U'$. If $(v, u_1) \in \mathcal{R}$, then v and x belong to the same connected component of $G' - \mathcal{X}$ and hence v and u belong to same connected component of $G - \mathcal{X}$. Similarly if $(v, u_1) \notin \mathcal{R}$, then v and x belong to the different connected components of $G' - \mathcal{X}$ and hence v and u belong to different connected components of $G - \mathcal{X}$. This concludes the proof of the lemma. $\square$

Lemma 10.13. *Let $\mathcal{I} = (G, T, \mathcal{R}, k, \ell)$ be an* MMCU *instance and let $uv \in E(G)$ be an edge with multiplicity more than $\ell + 1$. Then for any minimal solution $\mathcal{X} = (X, F)$ of $\mathcal{I}$, F does not contain any copies of uv.*

Proof. If $\{u, v\} \cap X \neq \emptyset$, then by Observation 10.1, we have that none of the copies of uv are in F. Otherwise, F contains at most ℓ copies of edge uv. Let $\mathcal{X}' = (X, F \setminus \{uv\})$. Then we have that for any two $x, y \in V(G)$, x and y are adjacent in $G - \mathcal{X}$ if and only if they are adjacent in $G - \mathcal{X}'$, contradicting the minimality of $\mathcal{X}$. $\square$

10.5.3 Borders and Recursive Understanding

In this section, we define the bordered problem and describe the recursive phase of the algorithm. Let $\mathcal{I} = (G, T, \mathcal{R}, k, \ell)$ be an MMCU instance and let $T_b \subseteq V(G) \setminus T$ be a set of border terminals, where $|T_b| \leq 2(k + \ell)$. Define $\mathcal{I}_b = (G, T, \mathcal{R}, k, \ell, T_b)$ to be an instance of the bordered problem. By $\mathbb{P}(\mathcal{I}_b)$ we define the set of all tuples $\mathcal{P} = (X_b, E_b, \mathcal{R}_b, k', \ell')$, such that $X_b \subseteq T_b$, E_b is an equivalence relation on $T_b \setminus X_b$, $\mathcal{R}_b$ is an equivalence relation on $T \cup (T_b \setminus X_b)$ such that $E_b \subseteq \mathcal{R}_b$ and $\mathcal{R}_b|_T = \mathcal{R}$, $k' \leq k$ and $\ell' \leq \ell$. For a tuple $\mathcal{P} = (X_b, E_b, \mathcal{R}_b, k', \ell')$, by $G_{\mathcal{P}}$ we denote the graph $G \cup E_b$, that is the graph G with additional edges E_b.

The intuition behind defining the tuple $\mathcal{P}$ is as following. The set X_b denotes the intersection of the solution with the border terminals. The equivalence relation E_b tells which of the border terminals can be connected from outside the graph considered. This can be looked at as analogous to *torso* operation on the graph. The equivalence relation $\mathcal{R}_b$ tells how the terminals and border terminals are going to get partitioned in different connected components after deleting the solution. Since deletion of any solution respects the relation $\mathcal{R}$, we have that $\mathcal{R}_b|_T = \mathcal{R}$. The numbers k' and ℓ' are guesses for how much the smaller graph is going to contribute to the solution.

We say that a tuple $\mathcal{X} = (X, F)$ is a solution to $(\mathcal{I}_b, \mathcal{P})$ where $\mathcal{P} = (X_b, E_b, \mathcal{R}_b, k', \ell')$ if $|X| \leq k'$, $|F| \leq \ell'$ and for all $u, v \in T \cup (T_b \setminus X_b)$, u and v belong to the same connected component of $G_{\mathcal{P}} - (X, F)$ if and only if $(u, v) \in \mathcal{R}_b$. We also say that $\mathcal{X}$ is a solution to $\mathcal{I}_b = (G, T, \mathcal{R}, k, \ell, T_b)$ whenever $\mathcal{X}$ is a solution to $\mathcal{I} = (G, T, \mathcal{R}, k, \ell)$. Now we define the bordered problem as follows.

Border-Mixed Multiway Cut-Uncut(B-MMCU) **Parameter(s):** k, ℓ
Input: An MMCU instance $\mathcal{I} = (G, T, \mathcal{R}, k, \ell)$ with G being connected and a set $T_b \subseteq V(G) \setminus T$ such that $|T_b| \leq 2(k + \ell)$; denote $\mathcal{I}_b = (G, T, \mathcal{R}, k, \ell, T_b)$.
Output: For each $\mathcal{P} = (X_b, E_b, \mathcal{R}_b, k', \ell') \in \mathbb{P}(\mathcal{I}_b)$, output a $\mathsf{sol}_{\mathcal{P}} = \mathcal{X}_{\mathcal{P}}$ being a minimal solution to $(\mathcal{I}_b, \mathcal{P})$, or $\mathsf{sol}_{\mathcal{P}} = \perp$ if no solution exists.

It is easy to see that MMCU reduces to B-MMCU, by putting $T_b = \emptyset$. Also, in this case, any answer to B-MMCU for $\mathcal{P} = (\emptyset, \emptyset, \mathcal{R}, k, \ell)$ returns a solution for MMCU instance. To bound the size of the solutions returned for an instance of B-MMCU we observe the following.

$$\begin{aligned}
|\mathbb{P}(\mathcal{I}_b)| &\leq (k+1)(\ell+1)(1 + |T_b|(|T_b| + (k+\ell)(k+\ell+1)))^{|T_b|} \\
&\leq (k+1)(\ell+1)(1 + 2(k+\ell)^2(k+\ell+3))^{2(k+\ell)} \\
&= 2^{\mathcal{O}((k+\ell)\log(k+\ell))}
\end{aligned}$$

This is true because $\mathcal{R}_b$ has at most $(k+\ell)(k+\ell+1) + |T_b|$ equivalence classes, E_b has at most T_b equivalence classes, each $v \in T_b$ can either go to X_b or choose an equivalence class in $\mathcal{R}_b$ and E_b, and k' and ℓ' have $k+1$ and $\ell+1$ possible values respectively. Let $q = (k+2\ell)(k+1)(\ell+1)(1+2(k+\ell)^2(k+\ell+3))^{2(k+\ell)} + k + \ell$, then all output solutions to a B-MMCU instance $\mathcal{I}_b$ affect at most $q - (k+\ell)$ vertices in total. Now we are ready to prove the lemma which is central for the recursive understanding step.

Lemma 10.14. *Assume we are given a* B-MMCU *instance* $\mathcal{I}_b = (G, T, \mathcal{R}, k, \ell, T_b)$ *and two disjoint sets of vertices* $Z, V^* \subseteq V(G)$*, such that* $|Z| \leq k+\ell$, $Z \cap T = \emptyset$, $Z_W := N_G(V^*) \subseteq Z$, $|V^* \cap T_b| \leq k + \ell$ *and the subgraph of* G *induced by* $W := V^* \cup Z_W$ *is connected. Denote* $G^* = G[W]$, $T_b^* = (T_b \cup Z_W) \cap W$, $T^* = T \cap W$, $\mathcal{R}^* = \mathcal{R}|_{T \cap W}$ *and* $\mathcal{I}^* = (G^*, T^*, R^*, k, \ell, T_b^*)$*. Then* $\mathcal{I}^*$ *is a proper* B-MMCU *instance. Moreover, if we denote by* $(\mathsf{sol}^*_{\mathcal{P}^*})_{\mathcal{P}^* \in \mathbb{P}(\mathcal{I}_b^*)}$ *an* *arbitrary output to the* B-MMCU *instance* $\mathcal{I}_b^*$ *and*

$$U(\mathcal{I}_b^*) = T_b^* \cup \{v \in V(G) \mid \mathcal{P}^* \in \mathbb{P}(\mathcal{I}_b^*), \mathsf{sol}_{\mathcal{P}^*}^* = \mathcal{X}_{\mathcal{P}^*}^* \neq \bot \textit{ and } \mathcal{X}_{\mathcal{P}^*}^* \textit{ affects } v\},$$

then there exists a correct output $(\mathsf{sol}_{\mathcal{P}})_{\mathcal{P} \in \mathbb{P}(\mathcal{I}_b)}$ *to the* B-MMCU *instance* $\mathcal{I}_b$ *such that whenever* $\mathsf{sol}_{\mathcal{P}} = \mathcal{X}_{\mathcal{P}} \neq \bot$ *and* $\mathcal{X}_{\mathcal{P}}$ *is a minimal solution to* $(\mathcal{I}_b, \mathcal{P})$ *then* $V(\mathcal{X}_{\mathcal{P}}) \cap V^* \subseteq U(\mathcal{I}_b^*)$.

Proof. To see that $\mathcal{I}_b^*$ is a proper B-MMCU instance, we observe that $G^* = G[W]$ is connected, $|Z_W| \leq |Z| \leq k + \ell$ and $|V^* \cap T_b| \leq k + \ell$, and hence $|T_b^*| \leq 2(k + \ell)$.

To show the second part of the lemma, let us assume $\mathcal{P} = (X_b, E_b, \mathcal{R}_b, k', \ell') \in \mathbb{P}(\mathcal{I}_b)$. Let $\mathcal{X}_{\mathcal{P}} = (X, F)$ be a solution to $(\mathcal{I}_b, \mathcal{P})$. We show that there exists another solution $\mathcal{X}'_{\mathcal{P}} = (X', F')$ to $(\mathcal{I}_b, \mathcal{P})$ such that $|X'| \leq |X|$, $|F'| \leq |F|$ and $V(\mathcal{X}'_{\mathcal{P}}) \cap V^* \subseteq U(\mathcal{I}_b^*)$. To that end, we define a tuple $\mathcal{P}^* = (X_b^*, E_b^*, \mathcal{R}_b^*, k^*, \ell^*)$ as follows. Let $F'' := F \setminus E(G[W])$, $X'' := X \setminus V^*$ and $\mathcal{X}''_{\mathcal{P}} = (X'', F'')$. Let $G'_{\mathcal{P}}$ be the graph such that $V(G'_{\mathcal{P}}) = V(G) \setminus W$ and $E(G'_{\mathcal{P}}) = E(G_{\mathcal{P}} - V^*) \setminus E(G_{\mathcal{P}}[Z_W])$. Observe that $V(G^*) \cap V(G'_{\mathcal{P}}) = Z_W$ and $E(G^*) \cap E(G'_{\mathcal{P}}) = \emptyset$.

- $X_b^* = X \cap T_b^*$.
- E_b^* is an equivalence relation on $T_b^* \setminus X_b^*$ such that $(u, v) \in E_b^*$ if and only if u and v are in the same connected component of $G'_{\mathcal{P}} - \mathcal{X}''_{\mathcal{P}}$.
- $\mathcal{R}_b^*$ is an equivalence relation on $T^* \cup (T_b^* \setminus X_b^*)$ such that $(u, v) \in \mathcal{R}_b^*$ if and only if u and v are in the same connected component of $G_{\mathcal{P}} - \mathcal{X}_{\mathcal{P}}$.
- $k^* := |X \cap W|$ and $\ell^* := |F \cap E(G[W])|$.

Since $|X| \leq k' \leq k$ and $|F| \leq \ell' \leq \ell$, we have that $k^* \leq k$ and $\ell^* \leq \ell$. Also, $E_b^* \subseteq \mathcal{R}_b^*$ because they are both corresponding to relation of being in the same connected component, but the graph considered for defining E_b^* is a subgraph of the one considered for defining $\mathcal{R}_b^*$. Hence we have that $\mathcal{P}^* \in \mathbb{P}(\mathcal{I}_b^*)$. We claim that $(X \cap W, F \cap E[(G[W]))$ is a solution to $(\mathcal{I}_b^*, \mathcal{P}^*)$. By definition, we have that $|X \cap W| \leq k^*$ and $|F \cap E[(G[W])]| \leq \ell^*$. Also, by definition of X_b^*, we have that $X \cap W \cap T_b^* = X_b^*$. Now look at two vertices

$u, v \in T^* \cup (T_b^* \setminus X_b^*)$. By definition, we have that $(u,v) \in \mathcal{R}_b^*$ if and only if there exists a path P connecting u and v in $G_{\mathcal{P}} - \mathcal{X}_{\mathcal{P}}$. We claim that such a path P exists if and only if there exists a path P^* connecting u and v in $G^*_{\mathcal{P}^*} - (X \cap W, F \cap E(G[W]))$. It is indeed so, because subpath of P with internal vertices in $V(G) \setminus W$ corresponds to an edge in E_b^* and vice versa. Thus, u and v are in the same connected component of $G^*_{\mathcal{P}^*} - (X \cap W, F \cap E[(G[W]))$ if and only if $(u,v) \in R_b^*$ and hence $(X \cap W, F \cap E[(G[W]))$ is a solution to $(\mathcal{I}_b^*, \mathcal{P}^*)$.

So we get that $\mathsf{sol}^*_{\mathcal{P}^*} = \mathcal{X}^*_{\mathcal{P}^*} =: (X^*, F^*) \neq \perp$ and $|X^*| \leq k^*$ and $|F^*| \leq \ell^*$. Let $\mathcal{X}'_{\mathcal{P}} = (X', F')$, where $X' = (X \setminus W) \cup X^*$ and $F' = (F \setminus E(G[W])) \cup F^*$. We claim that $\mathcal{X}'_{\mathcal{P}}$ is a solution for $(\mathcal{I}_b, \mathcal{P})$. As $|X^*| \leq |X \cap W|$ and $|F^*| \leq |F \cap E(G[W])|$, we have that $|X'| \leq |X| \leq k'$ and $|F'| \leq |F| \leq \ell'$. Also, since $X_b^* = X \cap T_b^*$ by definition of X_b^* and by properties of $\mathcal{X}^*_{\mathcal{P}^*}$, we have that $X_b^* = X^* \cap T_b^*$. Hence, $X \cap T_b^* = X^* \cap T_b^*$ and (X, F) is a solution for $(\mathcal{I}_b, \mathcal{P})$ which gives $X' \cap T_b = X_b$.

Now let us look at two vertices $u, v \in T \cup (T_b \setminus X_b)$. We need to show that u and v lie in the same connected component of $G_{\mathcal{P}} - \mathcal{X}'_{\mathcal{P}}$ if and only if they lie in the same connected component of $G_{\mathcal{P}} - \mathcal{X}_{\mathcal{P}}$. This will conclude the proof of the lemma.

Let $u, v \in T \cup (T_b \setminus X_b)$ lie in the same connected component of $G_{\mathcal{P}} - \mathcal{X}_{\mathcal{P}}$. Let $D := T \cup (T_b \setminus X_b) \cup Z_W$. Let P be a path connecting u and v in $G_{\mathcal{P}} - \mathcal{X}_{\mathcal{P}}$ and let $u = v_0, v_1, v_2, \ldots, v_r = v$ be the sequence of vertices that lie on P in that order such that $v_i \in D$ for all $i \in \{0, 1, \ldots, r\}$. Since $X \setminus V^* = X' \setminus V^*$ and $X \cap D = X' \cap D = X_b \cup X_b^*$, we have $v_i \notin X'$ for all $i \in \{0, 1, 2, \ldots, r\}$. Now, to finish this direction of the proof, we need to show that the vertices v_i and v_{i+1} lie in same connected component of $G_{\mathcal{P}} - \mathcal{X}'_{\mathcal{P}}$ for all $i \in \{0, 1, \ldots, r-1\}$.

Let P_i be the subpath of P between v_i and v_{i+1}. As $Z_W \subseteq D$, P_i is either a path in $G'_{\mathcal{P}}$ or a path in $G_{\mathcal{P}}[W]$. For the first case, we know that $X \setminus V^* = X' \setminus V^*$ and also we have that $F \setminus F^* = F' \setminus F^*$, and hence the path is also present on $G'_{\mathcal{P}} - \mathcal{X}_{\mathcal{P}}$ and hence in $G_P - \mathcal{X}'_{\mathcal{P}}$. In the second case, we have that $(v_i, v_{i+1}) \in \mathcal{R}_b^*$. As $(X' \cap W, F' \cap E(G[W])) = (X^*, F^*)$ is a solution to $(\mathcal{I}_b^*, \mathcal{P}^*)$, we infer that v_i and v_{i+1} are connected via a path P_i^* in $G^*_{\mathcal{P}^*} - (X' \cap W, F' \cap E(G[W]))$. Hence, u and v are connected in $G_{\mathcal{P}} - X'_{\mathcal{P}}$ if the path

P^* does not use any edge $w_1w_2 \in E_b^*$. By definition of E_b^*, $w_1w_2 \in E_b^*$ if and only if w_1 and w_2 lie in the same connected component of $G'_{\mathcal{P}} - (X'', F'')$. But we know that $X'' = X \setminus V^* = X' \setminus V^*$ and $F'' = F \setminus E(G[W]) = F' \setminus E(G[W])$ and hence w_1 and w_2 are also connected in $G'_{\mathcal{P}} - \mathcal{X}''_{\mathcal{P}}$ and hence in $G_{\mathcal{P}} - \mathcal{X}'_{\mathcal{P}}$. This completes one direction of the proof.

For the other direction, the proof is completely symmetrical and we omit the details. This completes the proof of the lemma. $\square$

Now we describe the recursive step of the algorithm.

Step 8. *Assume we are given a* B-MMCU *instance* $\mathcal{I}_b = (G, T, \mathcal{R}, k, \ell, T_b)$. *Invoke first the algorithm of Lemma 10.3 in a search for* $(q, k+\ell)$*-good node separation (with* $V^\infty = T$*). If it returns a good node separation* (Z, V_1, V_2)*, let* $j \in \{1, 2\}$ *be such that* $|V_j \cap T_b| \leq k + \ell$ *and denote* $Z^* = Z$*,* $V^* = V_j$*. Otherwise, if it returns that no such good node separation exists in* G*, invoke the algorithm of Lemma 10.4 in a search for* $(q, k+\ell)$*-flower separation w.r.t.* T_b *(with* $V^\infty = T$ *again). If it returns that no such flower separation exists in* G*, pass the instance* $\mathcal{I}_b$ *to the next step. Otherwise, if it returns a flower separation* $(Z, (V_i)_{i=1}^r)$*, denote* $Z^* = Z$ *and* $V^* = \bigcup_{i=1}^r V_i$*.*

In the case we have obtained Z^* *and* V^* *(either from Lemma 10.3 or Lemma 10.4), invoke the algorithm recursively for the* B-MMCU *instance* $\mathcal{I}_b^*$ *defined as in the statement of Lemma 10.14 for separator* Z^* *and set* V^**, obtaining an output* $(\mathsf{sol}^*_{\mathcal{P}^*})\mathcal{P}^* \in \mathbb{P}(\mathcal{I}_b^*)$. *Compute the set* $U(\mathcal{I}_b^*)$*. Bypass (in an arbitrary order) all vertices of* $V^* \setminus (T \cup U(\mathcal{I}_b^*))$*. Recall that* $T_b^* \subseteq U(\mathcal{I}_b^*)$*, so no border terminal gets bypassed. After all vertices of* $V^* \setminus U(\mathcal{I}_b^*)$ *are bypassed, perform the following operations on terminals of* $V^* \cap T$*:*

1. *As long as there exist two different* $u, v \in V^* \cap T$ *such that* $(u, v) \notin \mathcal{R}$*, and* $uv \in E(G)$*, then delete the edge* uv *and decrease* ℓ *by* 1*; if* ℓ *becomes negative by this operation, return* $\perp$ *for all* $\mathcal{P} \in \mathbb{P}(\mathcal{I}_b)$*.*

2. *As long as there exist two different* $u, v \in V^* \cap T$ *such that* $(u, v) \in \mathcal{R}$ *and either* $uv \in E(G)$ *or* $|N_G(u) \cap N_G(v)| > k + \ell$*, identify* u *and* v*.*

3. *If the above two rules are not applicable, then, as long as there exist three pairwise distinct terminals $u_1, u_2, \ldots, u_t \in T$ of the same equivalence class of $\mathcal{R}$ that have the same neighborhood and $t > \ell + 2$, delete u_i for $i > \ell + 2$ from the graph (and delete all pairs containing u_i from $\mathcal{R}$).*

Let $\mathcal{I}'_b$ be the outcome instance.

Finally, restart this step on the new instance $\mathcal{I}'_b$ and obtain a family of solutions $(\mathsf{sol}_{\mathcal{P}})_{\mathcal{P} \in \mathbb{P}(\mathcal{I}'_b)}$ and return this family as an output to the instance $\mathcal{I}_b$.

We first verify that the application of Lemma 10.14 is justified in Step 8. By definitions of good node separations and flower separations and by choice of V^*, we have that $|V^* \cap T_b| \leq k + \ell$ and $G[V^* \cup N_G(V^*)]$ is connected. Also, the recursive calls are applied to strictly smaller graphs because in good node separation, V_2 is deleted in the recursive call while in the other case, by definition of flower separation, we have that $Z \cup \bigcup_{i=1}^{r} V_i$ is a proper subset of $V(G)$.

After the bypassing operations, we have that V^* contains at most q vertices that are not terminals (at most $k + \ell$ border terminals and at most $q - (k + \ell)$ vertices which are neither terminals nor border terminals). Let us now bound the number of terminal vertices once Step 8 is applied. Note that, after Step 8 is applied, for any $v \in T \cap V^*$, we have $N_G(v) \subseteq (V^* \setminus T) \cup Z$ and $|(V^* \setminus T) \cup Z| \leq (q + k + \ell)$. Due to the first and second rule in Step 8, for any set $A \subseteq (V^* \setminus T) \cup Z$ of size $k + \ell + 1$, at most one terminal of $T \cap V^*$ is adjacent to all vertices of A. Due to the third rule in Step 8, for any set $B \subseteq (V^* \setminus T) \cup Z$ of size at most $k + \ell$ and for each equivalence class of $\mathcal{R}$, there are at most $\ell + 2$ terminals of this equivalence class with neighborhood exactly B. Let $q' := |T \cup V^*|$, then we have the following.

$$q' \leq (q + k + \ell)^{k+\ell+1} + (\ell + 2)(k + \ell)(k + \ell + 1) \sum_{i=1}^{k+\ell} (q + k + \ell)^i = 2^{\mathcal{O}((k+\ell)^2 \log(k+\ell))}$$

Lemma 10.15. *Assume that we are given a* B-MMCU *instance $\mathcal{I}_b = (G, T, \mathcal{R}, k, \ell, T_b)$ on which Step 8 is applied, and let $\mathcal{I}'_b$ be an instance after Step 8 is applied. Then any* *correct output to the instance $\mathcal{I}'_b$ is a correct output to the instance $\mathcal{I}_b$ as well. Moreover, if Step 8 outputs $\perp$ for all $\mathcal{P} \in \mathbb{P}(\mathcal{I}'_b)$, then this is a correct output to $\mathcal{I}_b$.*

Proof. We first note that by Lemma 10.14, for all $\mathcal{P} \in \mathbb{P}(\mathcal{I}_b)$, for all the vertices $v \notin U(\mathcal{I}^*_b)$, there exists a minimal solution to $(\mathcal{I}_b, \mathcal{P})$ that does not affect v, hence by Lemma 10.9, the bypassing operation is justified. The second and third rules are justified by Lemma 10.11 and Lemma 10.12 respectively. The first rule is justified by Lemma 10.10, and if application of this rule makes ℓ negative then for any $\mathcal{P} \in \mathbb{P}(\mathcal{I}_b)$, there is no solution to $(\mathcal{I}_b, \mathcal{P})$. □

Now we do a running time analysis for Step 8. The applications of Lemma 10.3 and Lemma 10.4 take time $\mathcal{O}(2^{\mathcal{O}(\mathsf{min}(q,k+\ell)\log(q+k+\ell))}n^3\log n) = \mathcal{O}(2^{\mathcal{O}((k+\ell)^2\log(k+\ell))}n^3\log n)$. Let $n' = |V^*|$; the recursive step is applied to a graph with at most $n'+k$ vertices and, after bypassing, there are at most $\mathsf{min}(n-1, n-n'+q+q')$ vertices left. Moreover, each bypassing operation takes $\mathcal{O}(n^2)$ time, the computation of $U(\mathcal{I}^*_b)$ takes $\mathcal{O}(2^{\mathcal{O}((k+\ell)\log(k+\ell))}n)$ time. Each application of Lemma 10.10 takes $\mathcal{O}(n^2)$ time and it can be applied at most ℓ times. So all applications of Lemma 10.10 take $\mathcal{O}(\ell n^2)$ time. Application of Lemma 10.11 takes $\mathcal{O}((k+\ell)n^2)$ time per operation, which can be implemented by having a counter for each pair of terminals and increasing those counters accordingly by considering every pair of terminals of $N_G(x)$, for each $x \in V(G) \setminus T$. Since when a counter reaches value $k+\ell+1$ for vertices u, v, we know that $|N_G(u) \cap N_G(v)| > k+\ell$, the total time consumed is bounded by $\mathcal{O}((k+\ell)n^2)$. Application of Lemma 10.12 takes $\mathcal{O}(n^2\log n)$ time per operation, since we can sort terminals from one equivalence class according to their sets of neighbours. Since both Lemma 10.11 and Lemma 10.12 decrease the number of terminals by at least 1, when applied, they can be applied at most n times. Hence, all applications of Lemma 10.11 and Lemma 10.12 takes total $\mathcal{O}(n^3(k+\ell+\log n))$ time. We also note that values of k and ℓ do not change during the recursive calls. So, we get the following recurrence relation for the running time of Step 8 as a function of vertices of the graph G.

$$T(n) \leq \mathsf{max}_{q+1\leq n'\leq n-q-1}\big(\mathcal{O}(2^{\mathcal{O}((k+\ell)^2\log(k+\ell))}n^3\log n)+T(n'+k+\ell)+T(\mathsf{min}(n-1, n-n'+q+q'))\big)$$

The base case for the recursive calls is the high connectivity phase, which takes time $\mathcal{O}(2^{\mathcal{O}((k+\ell)^3\log(k+\ell))}n^3\log n)$ as we will argue later. Solving the recurrence for the worst case gives $T(n) = \mathcal{O}(2^{\mathcal{O}((k+\ell)^3\log(k+\ell))}n^4\log n)$, which is the desired upper bound for the running time of the algorithm.

10.5.4 High Connectivity phase

In this section we describe the high connectivity phase for the algorithm. Assume we have a B-MMCU instance $\mathcal{I}_b = (G, T, \mathcal{R}, k', \ell', T_b)$ where Step 8 is not applicable. Let us fix $\mathcal{P} = (X_b, E_b, \mathcal{R}_b, k, \ell) \in \mathbb{P}(\mathcal{I}_b)$. We iterate through all possible values of $\mathcal{P}$ and try to find a minimal solution to $(\mathcal{I}_b, \mathcal{P})$. Since $|\mathbb{P}(\mathcal{I}_b)| = 2^{\mathcal{O}((k+\ell)\log(k+\ell))}$ it results in a factor of $2^{\mathcal{O}((k+\ell)\log(k+\ell))}$ in the running time. For a graph G, by $\mathcal{L}(G)$ we denote the set $V(G) \cup E(G)$. Similarly, for a tuple $\mathcal{X} = (X, F)$, by $\mathcal{L}(\mathcal{X})$ we denote the set $X \cup F$. We once again need to use lemmas 10.10-10.12 to bound number of terminals. We also need to apply Lemma 10.13 to bound the number of edges.

Step 9. *Apply Lemma 10.10, Lemma 10.11 and Lemma 10.12 exhaustively on the set T of terminals in the graph (as done in rules 1-3 of Step 8, but doing it for all of T instead of just $T \cap V^*$). Apply Lemma 10.13 to reduce multiplicity of all edges in the graph to at most $\ell + 1$.*

The running time analysis of applying lemmas 10.10-10.12 in this step is exactly the same as the one done in Step 8. Also, Lemma 10.13 can be applied in $\mathcal{O}(\ell n^2)$ time. Hence, the step takes $\mathcal{O}(n^3(k + \ell + \log n))$ time. After applying Step 9 exhaustively, we know that no two terminals are adjacent, and hence for any solution $\mathcal{X} = (X, F)$, we have that $F \cap E(G[T]) = \emptyset$.

Now we look at what can happen after deleting a set $\mathcal{X} = (X, F)$ from the graph G such that $X \subseteq V(G) \setminus T$, $F \subseteq E(G)$, $|X| \leq k$ and $|F| \leq \ell$. Since we have assumed that Step 8 is not applicable, for any $\mathcal{X} = (X, F)$ where $X \subseteq V(G) \setminus T$, $F \subseteq E(G)$, $|X| \leq k$ and $|F| \leq \ell$, Lemma 10.5 implies that the graph $G - \mathcal{X}$ contains at most $t := (2q + 2)(2(k + \ell) - 1) + 2(k + \ell) + 1$ connected components containing a non-terminal out of which at most one can contain more than q vertices outside T. Let us denote its vertex set by $\mathsf{big}(\mathcal{X})$ (observe that this can possibly be the empty set, in case such a component does not exist).

Now we define the notion of interrogating a solution, which will help us in highlighting the solution.

Definition 10.8. *Let $\mathcal{Z} = (Z, F')$ where $Z \subseteq V(G) \setminus T$, $F' \subseteq E(G) \setminus E(G[T])$, $|Z| \leq k$ and $|F'| \leq \ell$ and let $S \subseteq \mathcal{L}(G) \setminus T$. We say that S interrogates $\mathcal{Z}$ if the following holds:*

- *$S \cap \mathcal{L}(\mathcal{Z}) = \emptyset$;*
- *for any connected component C of $G - \mathcal{Z}$ with at most q vertices outside T, all vertices and edges of C belong to $S \cup T$.*

Lemma 10.16. *Let $q'' = (qt + k + \ell)^{k+\ell+1} + (\ell + 2)(k + \ell)(k + \ell + 1) \sum_{i=1}^{k+\ell} (qt + k + \ell)^i$. Let $\mathcal{F}$ be a family obtained by the algorithm of Lemma 10.5 for universe $U = \mathcal{L}(G) \setminus T$ and constants $a = qt + (\ell + 1)\binom{q''+qt}{2}$ and $b = k + \ell$, Then, for any $\mathcal{Z} = (Z, F')$ where $Z \subseteq V(G) \setminus T$, $F' \subseteq E(G) \setminus E(G[T])$, $|Z| \leq k$ and $|F'| \leq \ell$, there exists a set $S \in \mathcal{F}$ that interrogates $\mathcal{Z}$.*

Proof. Let $\mathcal{Z} = (Z, F')$ where $Z \subseteq V(G) \setminus T$, $F' \subseteq E(G) \setminus E(G[T])$, $|Z| \leq k$ and $|F'| \leq \ell$. Let A be the union of vertex sets of all connected components of $G - \mathcal{Z}$ that have at most q vertices outside T and let B be the set of edges of these connected components. By Lemma 10.5, $|A \setminus T| \leq qt$. Also, since we have applied Step 9 exhaustively, we have that no two terminals are adjacent in the graph. Let $C = \{u \mid uv \in E(G) \cap F', v \in A \cap T\}$. Clearly, $C \cap T = \emptyset$, $|C| \leq \ell$ and for any $v \in A$, $N(v) \subseteq (A \setminus T) \cup Z \cup C$ which gives $|N(A \cap T)| \leq |(A \setminus T) \cup Z \cup C| \leq qt + k + \ell$. Now doing the same analysis as in Step 8 gives us $|A \cap T| \leq q''$. Since the multiplicity of all the edges is bounded by $\ell + 1$, we have that $|(A \setminus T) \cup B| \leq qt + (\ell + 1)\binom{q''+qt}{2}$ and $|\mathcal{L}(\mathcal{Z})| \leq k + \ell$. So, by Lemma 10.2, there is a set $S \in \mathcal{F}$ that is disjoint with $\mathcal{L}(\mathcal{Z})$ and contains $(A \setminus T) \cup B$. By construction of A and B, S interrogates $\mathcal{Z}$. This completes the proof of the lemma. □

Step 10. *Compute the family $\mathcal{F}$ from Lemma 10.16 and branch into $|\mathcal{F}|$ subcases, indexed by sets $S \in \mathcal{F}$. In a branch S we seek for a minimal solution $\mathcal{X}_{\mathcal{P}}$ to $(\mathcal{I}_b, \mathcal{P})$, which is interrogated by S.*

Note that since we have $q'' = 2^{\mathcal{O}((k+\ell)^2 \log(k+\ell))}$ and $q, t = 2^{\mathcal{O}((k+\ell)\log(k+\ell))}$, the family $\mathcal{F}$ of Lemma 10.2 is of size $\mathcal{O}(2^{\mathcal{O}((k+\ell)^3 \log(k+\ell))} \log n)$ and can be computed in $\mathcal{O}(2^{\mathcal{O}((k+\ell)^3 \log(k+\ell))} n \log n)$ time. The correctness of Step 10 is obvious from Lemma 10.16. As discussed, it can be applied in $\mathcal{O}(2^{\mathcal{O}((k+\ell)^3 \log(k+\ell))} n \log n)$ time and gives rise to $\mathcal{O}(2^{\mathcal{O}((k+\ell)^3 \log(k+\ell))} \log n)$ subcases.

Lemma 10.17. *Let $\mathcal{X}_{\mathcal{P}} = (X, F)$ be a solution to $(\mathcal{I}_b, \mathcal{P})$ interrogated by S. Then there exists a set $T^{\mathsf{big}} \subseteq T \cup (T_b \setminus X_b)$ that is empty or contains all vertices of exactly one equivalence class of $\mathcal{R}_b$, such that $X \subseteq (X_b \cup N_G(S(T^{\mathsf{big}}))$ and $F = A_{G,X}(S(T^{\mathsf{big}}))$, where $S(T^{\mathsf{big}})$ is the union of vertex sets of all connected components of $G(S \cup T \cup (T_b \setminus X_b))$ that contain a vertex of $(T \cup (T_b \setminus X_b)) \setminus T^{\mathsf{big}}$ and $A_{G,X}(S(T^{\mathsf{big}}))$ is set of edges in G which have at least one end point in $S(T^{\mathsf{big}})$ but do not belong to any of the connected components of $G[S(T^{\mathsf{big}})]$ and are not incident on X.*

Proof. Let $\mathcal{X}_{\mathcal{P}} = (X, F)$. Consider the graph $G_{\mathcal{P}} - \mathcal{X}_{\mathcal{P}}$ and let $\mathsf{big}_{\mathcal{P}}(\mathcal{X}_{\mathcal{P}})$ be the vertex set of the connected component of $G_{\mathcal{P}} - \mathcal{X}_{\mathcal{P}}$ that contains $\mathsf{big}(\mathcal{X}_{\mathcal{P}})$ (recall that $G_{\mathcal{P}}$ is the graph G with additional edges E_b ; thus $\mathsf{big}_{\mathcal{P}}(\mathcal{X}_{\mathcal{P}})$ may be significantly larger than $\mathsf{big}(\mathcal{X}_{\mathcal{P}})$). If $\mathsf{big}(\mathcal{X}_{\mathcal{P}}) = \emptyset$, then we take $\mathsf{big}_{\mathcal{P}}(\mathcal{X}_{\mathcal{P}})$ to be the empty set as well. As $\mathcal{X}_{\mathcal{P}}$ is a solution to $(\mathcal{I}_b, \mathcal{P})$, we have $X \cap T_b = X_b$. Define $T^{\mathsf{big}} = (T \cup (T_b \setminus X_b)) \cap \mathsf{big}_{\mathcal{P}}(\mathcal{X}_{\mathcal{P}})$; as $\mathcal{X}_{\mathcal{P}}$ is a solution to $(\mathcal{I}_b, \mathcal{P})$, T^{big} is empty or contains vertices of exactly one equivalence class of R_b.

Now let C be the vertex set of a connected component of $G - \mathcal{X}_{\mathcal{P}}$ that contains a vertex $v \in (T \cup (T_b \setminus X_b)) \setminus T^{\mathsf{big}}$. Clearly, $v \notin \mathsf{big}_{\mathcal{P}}(\mathcal{X}_{\mathcal{P}})$. As S interrogates $\mathcal{X}_{\mathcal{P}}$, $\mathsf{big}_{\mathcal{P}}(\mathcal{X}_{\mathcal{P}})$ contains $\mathsf{big}(\mathcal{X}_{\mathcal{P}})$ and $X \cap (T \cup T_b) = X_b \subseteq T_b$, we infer that C is the vertex set of a connected component of $G(S \cup T \cup (T_b \setminus X_b))$ as well. As $v \in C$, C is a connected component of $G[S(T^{\mathsf{big}})]$. Let $X' = X_b \cup (X \cap N_G(S(T^{\mathsf{big}})))$ and let $F' = A_{G,X'}(S(T^{\mathsf{big}})) \cap F$. We claim that $\mathcal{X}' = (X', F')$ is a solution to $(\mathcal{I}_b, \mathcal{P})$.

As $X_b \subseteq X$, we have that $X' \subseteq X$ and hence $|X'| \leq k$. Also, since $F' \subseteq F$, we have that $|F'| \leq \ell$. Moreover, as $X' \subseteq X$, $F' \subseteq F$ and $\mathcal{X}_{\mathcal{P}} = (X, F)$ is a solution to $(\mathcal{I}_b, \mathcal{P})$, if $(u, v) \in \mathcal{R}_b$ then u and v are in the same connected component of $G_{\mathcal{P}} - \mathcal{X}'_{\mathcal{P}}$. We now show that for any $(u, v) \notin \mathcal{R}_b$ the vertices u and v are in different connected components of $G_{\mathcal{P}} - \mathcal{X}'_{\mathcal{P}}$. For the sake of contradiction, let $u, v \in T \cup (T_b \setminus X_b)$ such that $(u, v) \notin \mathcal{R}_b$, u

and v lie in the same connected component of $G_{\mathcal{P}} - \mathcal{X}'_{\mathcal{P}}$ and distance between u and v is minimum possible. Let P be the shortest path between u and v in $G_{\mathcal{P}} - \mathcal{X}'_{\mathcal{P}}$.

As $\mathcal{X}_{\mathcal{P}} = (X, F)$ is a solution to $(\mathcal{I}_b, \mathcal{P})$, u and v lie in different connected components of $G_{\mathcal{P}} - \mathcal{X}_{\mathcal{P}}$. Without loss of generality, let $v \notin T^{\mathsf{big}}$ and let C be the connected component of $G - \mathcal{X}_{\mathcal{P}}$ that contains v. Since $(u, v) \notin \mathcal{R}_b$ and $G - \mathcal{X}_{\mathcal{P}}$ is a subgraph of $G_{\mathcal{P}} - \mathcal{X}_{\mathcal{P}}$, we have that $u \notin C$. Since for every edge xy in G such that $x \in C$ and $y \notin C$, we have that either $y \in X'$ or $xy \in F'$, the path P contains an edge $v_1u_1 \in E_b$ such that $v_1 \in C$ and $u_1 \notin C$. So in that case we have that $v_1, u_1 \in T_b$ and hence $(v_1, u_1) \in \mathcal{R}_b$. Also, since $v_1 \in C$, $(v, v_1) \in \mathcal{R}_b$ and since $\mathcal{R}_b$ is an equivalence relation, we have that $(v, u_1) \in \mathcal{R}_b$. But by our assumption, $(u, v) \notin \mathcal{R}_b$, which gives that $(u_1, u) \notin \mathcal{R}_b$. Hence $u_1, u \in T \cup (T_b \setminus X_b)$ such that $(u_1, u) \notin \mathcal{R}_b$ and they are connected by a proper subpath of P in $G_{\mathcal{P}} - \mathcal{X}'_{\mathcal{P}}$, which contradicts our choice of u, v and P. This completes the proof of the lemma. □

Now we are ready to give the final step of the algorithm. The correctness of the step follows from Lemma 10.17 and the fact that if S interrogates a solution $\mathcal{X}$ to $(\mathcal{I}_b, \mathcal{P})$, then $|N_G(S(T^{\mathsf{big}}))| \leq k + \ell$.

Step 11. *For each branch, where S is the corresponding guess, we do the following. For each set T^{big} that is empty or contains all vertices of one equivalence class of $\mathcal{R}_b$, if $|N_G(S(T^{\mathsf{big}}))| \leq k + \ell$, then for each $X \subseteq X_b \cup N_G(S(T^{\mathsf{big}}))$ such that $|X| \leq k$, and $F = A_{G,X}(S(T^{\mathsf{big}}))$, check whether (X, F) is a solution to $(\mathcal{I}_b, \mathcal{P})$ interrogated by S. For each $\mathcal{P}$, output a minimal solution to $(\mathcal{I}_b, \mathcal{P})$ that is interrogated by S. Output $\perp$ if no solution is found for any choice of S, T^{big} and X.*

Note that $\mathcal{R}$ has at most $(k + \ell)(k + \ell + 1)$ equivalence classes. As $|T_b| \leq 2(k + \ell)$, we have $\mathcal{R}_b$ has at most $(k + \ell)(k + \ell + 3)$ equivalence classes, and hence there are at most $(k + \ell)(k + \ell + 3) + 1$ choices of the set T^{big}. For each T^{big}, computing $N_G(S(T^{\mathsf{big}}))$ and checking whether $|N_G(S(T^{\mathsf{big}}))| \leq k + \ell$ takes $\mathcal{O}(n^2)$ time. Since $X_b \leq k$, there are at most $(k+1)(2k+\ell)^k$ choices for X, and then computing $F = A_{G,X}(S(T^{\mathsf{big}}))$ and checking whether (X, F) is a solution to $(\mathcal{I}_b, \mathcal{P})$ interrogated by S take $\mathcal{O}(n^2)$ time each. Finally, checking whether the solution is minimal or not and computing a minimal solution takes additional $\mathcal{O}((k + \ell)n^2)$ time. Therefore Step 11 takes $\mathcal{O}(2^{\mathcal{O}((k+\ell)^3 \log(k+\ell))} n^2 \log n)$ time for all subcases.

This finishes the description of fixed-parameter algorithm for MMCU and we get the following theorem.

Theorem 10.9. MMCU *can be solved in* $\mathcal{O}(2^{\mathcal{O}((k+\ell)^3 \log(k+\ell))} n^4 \log n)$ *time.*

10.6 Algorithm for MMCU with undeletable vertices

In this section, we describe an algorithm for MMCU*. To solve this problem, we will use the algorithm to solve MMCU which outputs a minmial solution. Let us first recall the definition of MMCU* from Chapter 9.

MMCU* **Parameter(s):** k,ℓ

Input: A graph G, integers k and ℓ, $T \subseteq V(G)$, an equivalence relation $\mathcal{R}$ on T having at most two equivalence classes, and set of undeletable vertices $U \subseteq V(G)$.

Output: Output a minimal solution $\mathcal{X} = (X, F)$ to MMCU instance $(G, T, \mathcal{R}, k, \ell)$ such that $X \cap U = \emptyset$ and $\perp$ if no such solution exists.

The definitions of a solution and minimal solutions to an MMCU*instance are exactly the same as in the case of MMCU. Observe that MMCU* is essentially an instance of MMCU, where in addition to an instance $(G, k, \ell, T, \mathcal{R})$ of MMCU, we are also provided with a set of undeletable vertices $U \subseteq V(G)$, and we require the vertices in the solution to be disjoint from U. We describe the algorithm for the case when number of equivalence classes in $\mathcal{R}$ is at most two, as that is what we needed for our application to Pseudobipartite Deletion. But we remark that the reduction can be made to work for unbounded number of equivalence classes as well, because after applying some preprocessing, as explained in [RRS16], the number of equivalence classes for any connected component can be bounded.

Now, we show how to reduce an instance of MMCU* to bounded number of instance of MMCU. For that, we basically guess how the solution is going to partition the vertices of U. We first prove the following lemma.

Lemma 10.18. *Let* $(G, k, \ell, T, \mathcal{R}, U)$ *be an instance of* MMCU* *and let* $\mathbb{R}$ *be the set of* *equivalence relations $\mathcal{R}_U$ on $T \cup U$ such that $\mathcal{R}_U|_T = \mathcal{R}$. Then there exists $\mathcal{R}_U^* \in \mathbb{R}$ such that the solution to* MMCU *instance $(G, k, \ell, T \cup U, \mathcal{R}_U^*)$ is a solution to* MMCU* *instance $(G, k, \ell, T, \mathcal{R}, U)$ as well.*

Proof. First we observe that for any $\mathcal{R}_U \in \mathbb{R}$, since $\mathcal{R}_U|_T = \mathcal{R}$, for any solution $\mathcal{X}'$ to MMCU instance $(G, k, \ell, T \cup U, \mathcal{R}_U)$, $u, v \in T$ belong to the same connected component of $G - \mathcal{X}'$ if and only if $(u, v \in \mathcal{R})$. Let $\mathcal{X}$ be a solution to MMCU* instance $(G, k, \ell, T, \mathcal{R}, U)$. Now we just need to show that there exists $\mathcal{R}_U^* \in \mathbb{R}$ such that for the solution $\mathcal{X}'$ to the MMCU instance $(G, k, \ell, T \cup U, \mathcal{R}_U^*)$, we have that $\mathcal{X}' \leq \mathcal{X}$. We show that $\mathcal{X}$ itself is such a solution. We define $\mathcal{R}_U^*$ as the equivalence relation on $T \cup U$ such that $(u, v) \in \mathcal{R}_U^*$ if and only if u and v lie in the same connected component of $G - \mathcal{X}$. As $\mathcal{X}$ was a solution to MMCU* instance $(G, k, \ell, T, \mathcal{R}, U)$, we have that $\mathcal{R}_U^*|_T = \mathcal{R}$. Also, by definition of $\mathcal{R}_U^*$, for all $u, v \in T \cup U$, u and v belong to the same connected component of $G - \mathcal{X}$ if and only if $(u, v) \in \mathcal{R}_U^*$. Hence, $\mathcal{X}$ is a solution to MMCU instance $(G, k, \ell, T \cup U,\ R_U^*)$. ☐

Now, to solve an instance $(G, k, \ell, T, \mathcal{R}, U)$ of MMCU*, for all equivalence relations $\mathcal{R}_U$ on $T \cup U$ such that $\mathcal{R}_U|_T = \mathcal{R}$, we solve the MMCU instance $(G, k, \ell, T \cup U, \mathcal{R}_U)$. Let $\mathcal{X}_{\mathcal{R}_U}$ be the solution for the MMCU instance $(G, k, \ell, T \cup U, \mathcal{R}_U)$. We return the solution $\mathcal{X}_{\mathcal{R}_U^*}$ such that for any $\mathcal{X}_{\mathcal{R}_U}$ such that $\mathcal{R}_U^* \neq \mathcal{R}_U$, we have that $\mathcal{X}_{\mathcal{R}_U^*} \leq \mathcal{X}_{\mathcal{R}_U}$. The correctness of the algorithm follows from Lemma 10.18. Since the number of equivalence classes in $\mathcal{R}$ is at most two, the number of equivalence classes in any equivalence relation $\mathcal{R}_U$ on $T \cup U$ such that $\mathcal{R}_U|_T = \mathcal{R}$ is at most $|U| + 2$. So, the number of such equivalence relations is bounded by $(|U| + 2)^{|U|}$. Hence, we can solve an instance of MMCU* by making at most $(|U| + 2)^{|U|}$ calls to the algorithm for MMCU given in Theorem 10.9. which takes $\mathcal{O}(2^{\mathcal{O}((k+\ell)^3 \log(k+\ell))} n^4 \log n)$ time. Hence we get the following theorem.

Theorem 10.10. MMCU* *can be solved in $(|U| + 2)^{|U|} \mathcal{O}(2^{\mathcal{O}((k+\ell)^3 \log(k+\ell))} n^4 \log n)$ time.*

Part IV

Conclusion

Chapter 11

Conclusion and Future Directions

The aim of this thesis was to study graph editing problems from various aspects. Many of the graph editing problems including, but not limited to Vertex Cover, Clustering, Feedback Vertex Set, are not only important from the algorithmic and applications point of view, but also the research on them has led to deep insights into the theoretical world. As most of these problems turn out to be NP–complete, parameterized complexity is one of the natural ways to get tractable algorithms for them. In this thesis, we studied many variations of graph editing problems and obtained FPT algorithms and kernels for the same. We also give kernelization lower bounds for some induced subgraph problems showing that they are not likely to have polynomial kernels. The results in the thesis are summarized in the next section, and in the final section, we discuss some future directions of research.

11.1 Results in the thesis

We first mention the directions of research pursued in the thesis.

- We looked for applications of conondeterminism in composition for proving lower bounds for kernelization. This was one of very few lower bound results which used the aspect of conondeterminism allowed by the framework.

- We introduced the notion of coNP reductions, which can be useful in proving many lower bounds on kernelization, as it does away with the requirement of proving imrovement versions to be NP-hard for most problems.
- We initiated study of finding induced subgraph problems on restricted graph classes which are W[1] hard on general graphs.
- We looked at generalizations of Feedback Vertex Set and Odd Cycle Transversal problems and obtained FPT and kernelization results for the same.
- We generalized the classic Minimum Cut problem to include both vertices and edges, showed it to be NP–complete and arrived at FPT results.

Now we summarize the results in the thesis.

- Π-Induced Subgraph does not admit a polynomial kernel unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$ for Π being any of the following classes of graphs: Perfect graphs, Chordal graphs, Interval graphs, Cluster graphs, Weakly Chordal graphs, Cographs, Comparability graphs, Proper Interval graphs, Split graphs and Claw-free graphs (Chapter 4).
- p-Max q-Colorable Induced Subgraph is FPT on Perfect graphs and Co-chordal graphs. Also, on perfect graphs, the problem does not admit any polynomial kernel for any fixed $q \geq 2$ (Chapter 5).
- Split Vertex Deletion can be solved in time $\mathcal{O}^*(2^k)$ and admits a kernel of size $\mathcal{O}(k^3)$. Split Edge Deletion can be solved in time $\mathcal{O}^*(2^{\sqrt{k}\log k})$ time and admits a kernel of size $\mathcal{O}(k^2)$ (Chapter 6).
- Almost Forest Deletion is FPT and can be solved in time $\mathcal{O}^*(5.0024^(k+\ell))$ and admits a kernel of size $\mathcal{O}(k\ell(k+\ell))$. Moreover, it can be solved on planar graphs in $\mathcal{O}^*(c^{\sqrt{k+\ell}})$ time (Chapter 7).
- ℓ-pseudoforest Deletionis FPT and can be solved in time $\mathcal{O}^*(c_\ell^k)$ time where the constant c_ℓ depends only on ℓ. It also admits a kernel of size $c'_\ell k^2$, where c'_ℓ is another constant depending only on ℓ and hence a uniform kernel of size k^2 for every

fixed ℓ. Pseudoforest Deletioncan be solved in time $\mathcal{O}^*(7.5618^k)$ and admits a kernel of size $\mathcal{O}(k^2)$ (Chapter 8).

- Strong Bipartite Deletion is FPT(Chapter 9).
- Mixed Multiway Cut-Uncut is FPT(Chapter 10).

11.2 Future Directions

Now we mention some potential areas of research this thesis points towards.

- The result of Khot and Raman [KR02] characterizes the fixed parameter tractability of Π-Induced Subgraph where Π is any non-trivial hereditary graph class. Even though our results resolve the question of polynomial for most well known graph classes, getting a similar result which characterizes kernelization complexity of Π-Induced Subgraph for all non-trivial hereditary graph classes remains elusive and would be very interesting.
- We initiated study of Π-Induced Subgraph on restricted classes of graphs in this thesis. It would be interesting to catalogue other problems which are W-hard on general graphs but become FPT on some restricted class.
- We saw that the combination of iterative compression technique and recursive understanding technique introduced by Chitnis et al. [CCH+12] can be useful in solving graph modification problems. We believe that the technique is very general and that full power of it is yet to be utilized.
- The Strong $\mathcal{F}$ Deletion problem introduced in the thesis opens many new interesting areas. As we have seen, these problems are significantly harder than normal graph deletion problems, but in the case of $\mathcal{F}$ being bipartite graphs, we were able to arrive at FPT results. It is an interesting question to ask whether there are any other graph classes $\mathcal{F}$ for which we can show the Strong $\mathcal{F}$ Deletion problem to be FPT.

Bibliography

[ABKK$^+$10] Louigi Addario-Berry, W. Sean Kennedy, Andrew D. King, Zhentao Li, and Bruce A. Reed. Finding a maximum-weight induced k-partite subgraph of an i-triangulated graph. *Discrete Applied Mathematics*, 158(7):765–770, 2010. 7, 56

[AK10] Faisal Abu-Khzam. A kernelization algorithm for d-hitting set. *Journal of Computer and System Sciences*, 76(7):524–531, 2010. 9, 80

[ALMS16] Akanksha Agrawal, Daniel Lokshtanov, Amer E. Mouawad, and Saket Saurabh. Simultaneous Feedback Vertex Set: A Parameterized Perspective. In Nicolas Ollinger and Heribert Vollmer, editors, *33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016)*, volume 47 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:15, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. 166

[ALS09] Noga Alon, Daniel Lokshtanov, and Saket Saurabh. Fast fast. In *ICALP (1)*, pages 49–58, 2009. 9, 81, 91, 92, 93

[APS01] Noga Alon, János Pach, and József Solymosi. Ramsey-type theorems with forbidden subgraphs. *Combinatorica*, 21(2):155–170, 2001. 35

[AS14] Nicola Apollonio and Bruno Simeone. The maximum vertex coverage problem on bipartite graphs. *Discrete Applied Mathematics*, 165:37–48, 2014. 14, 207, 211

[BDFH09] 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. 25, 31, 33

[BDT11] Nicolas Bousquet, Jean Daligault, and Stéphan Thomassé. Multicut is FPT. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 459–468, 2011. 206

[BFL$^+$09] Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (meta) kernelization. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 629–638, 2009. 136

[BHKK07] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets möbius: fast subset convolution. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 67–74, 2007. 61

[BJK14] 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. 26, 31

[BTY11] Hans L. Bodlaender, Stéphan Thomassé, and Anders Yeo. Kernel bounds for disjoint cycles and disjoint paths. *Theoretical Computer Science*, 412(35):4570–4578, 2011. 28

[BW13] Lowell W Beineke and Robin J Wilson. Topics in structural graph theory. 2013. 207

[Cai96] Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996. 4, 34, 79, 80

[Cai03] Leizhen Cai. Parameterized complexity of vertex colouring. *Discrete Applied Mathematics*, 127(3):415–429, 2003. 166

[Cao15] Yixin Cao. Unit interval editing is fixed-parameter tractable. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 306–317. Springer, 2015. 166

[Cao16] Yixin Cao. Linear recognition of almost interval graphs. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1096–1115, 2016. 166

[CCH$^+$12] Rajesh Hemant Chitnis, Marek Cygan, MohammadTaghi Hajiaghayi, Marcin Pilipczuk, and Michal Pilipczuk. Designing FPT algorithms for cut problems using randomized contractions. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 460–469, 2012. 12, 15, 169, 170, 171, 172, 173, 206, 207, 208, 209, 210, 211, 212, 237

[CFK$^+$15] Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer-Verlag, 2015. 9

[CHM13] Rajesh Hemant Chitnis, MohammadTaghi Hajiaghayi, and Dániel Marx. Fixed-parameter tractability of directed multiway cut parameterized by the size of the cutset. *SIAM J. Comput.*, 42(4):1674–1696, 2013. 206

[CJ03] Liming Cai and David Juedes. On the existence of subexponential parameterized algorithms. *J. Comput. Syst. Sci.*, 67:789–807, December 2003. 85

[CLL$^+$08] Jianer Chen, Yang Liu, Songjian Lu, Barry O'Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5), 2008. 207

[CLL09] Jianer Chen, Yang Liu, and Songjian Lu. An improved parameterized algorithm for the minimum node multiway cut problem. *Algorithmica*, 55(1):1–13, 2009. 206

[CM14] Yixin Cao and Dániel Marx. Chordal editing is fixed-parameter tractable. In *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014), STACS 2014, March 5-8, 2014, Lyon, France*, volume 25 of *LIPIcs*, pages 214–225. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014. 166, 207

[CM15] Yixin Cao and Dániel Marx. Interval deletion is fixed-parameter tractable. *ACM Transactions on Algorithms*, 11(3):21:1–21:35, 2015. 166

[CNP$^+$11] 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 *FOCS*, pages 150–159, 2011. 101

[CP13] Marek Cygan and Marcin Pilipczuk. Split vertex deletion meets vertex cover: New fixed-parameter and exact exponential-time algorithms. *Inf. Process. Lett.*, 113(5-6):179–182, 2013. 9, 81, 82, 99

[CSRL01] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001. 17

[Die05] Reinhard Diestel. *Graph Theory*. Springer, 2005. 35

[DLMR10] Konrad Dabrowski, Vadim V. Lozin, Haiko Müller, and Dieter Rautenbach. Parameterized algorithms for the independent set problem in some hereditary graph classes. In *IWOCA*, volume 6460 of *Lecture Notes in Computer Science*, pages 1–9. Springer-Verlag, 2010. 7, 56

[DLS09] Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Incompressibility through colors and ids. In *ICALP (1)*, volume 5555 of *Lecture Notes in Computer Science*, pages 378–389, 2009. 31, 74

[DM12] Holger Dell and Dániel Marx. Kernelization of packing problems. In *SODA*, pages 68–81, 2012. 31

[DvM10] Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. In *STOC*, pages 251–260, 2010. 31

[EH89] Paul Erdős and András Hajnal. Ramsey-type theorems. *Discrete Applied Mathematics*, 25(1-2):37–52, 1989. 34

[Erd47] Paul Erdős. Some remarks on the theory of graphs. *Bulletin of the American Mathematical Society*, 53:292–294, 1947. 34

[ES35] Paus Erdős and George Szekeres. A combinatorial problem in geometry. *Compositio Mathematica*, 2:463–470, 1935. 33

[FH77] S. Foldes and P.L. Hammer. Split graphs. Congressus Numerantium 19, pages 311–315, 1977. 79, 80

[FJR13] 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. 167

[FKP$^+$11] Fedor V. Fomin, Stefan Kratsch, Marcin Pilipczuk, Michal Pilipczuk, and Yngve Villanger. Subexponential fixed-parameter tractability of cluster editing. *CoRR*, abs/1112.4419, 2011. 81

[FLM$^+$11] Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, Geevarghese Philip, and Saket Saurabh. Hitting forbidden minors: Approximation and kernelization. In *Proceedings of the 28th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 9 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 189–200. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011. 116, 158

[FLM$^+$15] Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, M. S. Ramanujan, and Saket Saurabh. Solving d-sat via backdoors to small treewidth. In *SODA*, pages 630–641, 2015. 139

[FLMS12] Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar F-deletion: Approximation, kernelization and optimal FPT algorithms.

In *Proceedings of the 53rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 470–479. IEEE, 2012. 11, 130, 139, 165, 166, 167

[FLS14] Fedor V. Fomin, Daniel Lokshtanov, and Saket Saurabh. Efficient computation of representative sets with applications in parameterized and exact algorithms. In *SODA*, pages 142–151, 2014. 8, 57, 58, 66, 121, 122

[Fra12] András Frank. Connections in combinatorial optimization. *Discrete Applied Mathematics*, 160(12):1875, 2012. 207

[FS11] Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *Journal of Computer and System Sciences*, 77(1):91–106, 2011. 25, 28, 31

[FSV13] Fedor V. Fomin, Saket Saurabh, and Yngve Villanger. A polynomial kernel for proper interval vertex deletion. *SIAM J. Discrete Math.*, 27(4):1964–1976, 2013. 34

[Fuj98] Toshihiro Fujito. A unified approximation algorithm for node-deletion problems. *Discrete Appl. Math.*, 86:213–231, September 1998. 79, 165

[FV12] Fedor V. Fomin and Yngve Villanger. Subexponential parameterized algorithm for minimum fill-in. In *SODA*, pages 1737–1746, 2012. 9, 79, 81

[Gal64] T. Gallai. Maximum-minimum sätze und verallgemeinerte faktoren von graphen. *Acta Mathematica Academiae Scientiarum Hungarica*, 12(1-2):131–173, 1964. 113, 155

[GGH$^+$06] Jiong Guo, Jens Gramm, Falk Hüffner, Rolf Niedermeier, and Sebastian Wernicke. Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. *J. Comput. Syst. Sci.*, 72(8):1386–1396, 2006. 192

[GJLS15] Archontia C. Giannopoulou, Bart M. P. Jansen, Daniel Lokshtanov, and Saket Saurabh. Uniform kernelization complexity of hitting forbidden minors. *CoRR*, abs/1502.03965, 2015. to appear in ICALP 2015. 130

[GKK$^+$15] Esha Ghosh, Sudeshna Kolay, Mrinal Kumar, Pranabendu Misra, Fahad Panolan, Ashutosh Rai, and M. S. Ramanujan. Faster parameterized algorithms for deletion to split graphs. *Algorithmica*, 71(4):989–1006, 2015. 9

[GKMW11] Martin Grohe, Ken-ichi Kawarabayashi, Dániel Marx, and Paul Wollan. Finding topological subgraphs is fixed-parameter tractable. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 479–488, 2011. 15, 169

[Gol80] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, 1980. 35, 82

[GRS16] Robert Ganian, M. S. Ramanujan, and Stefan Szeider. Discovering archipelagos of tractability for constraint satisfaction and counting. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1670–1681, 2016. 168

[GT12] Qian-Ping Gu and Hisao Tamaki. Improved bounds on the planar branchwidth with respect to the largest grid minor size. *Algorithmica*, 64(3):416–453, 2012. 127

[Guo07] Jiong Guo. Problem kernels for NP-complete edge deletion problems: Split and related graphs. In *Proceedings of the 18th International Symposium on Algorithms and Computation (ISAAC)*, volume 4835 of *Lecture Notes in Comput. Sci.*, pages 915–926. Springer, 2007. 9, 79, 81, 94

[HM09] Pinar Heggernes and Federico Mancini. Minimal split completions. *Discrete Applied Mathematics*, 157(12):2659–2669, 2009. 81

[HN10] Danny Harnik and Moni Naor. On the compressibility of NP instances and cryptographic applications. *SIAM Journal on Computing*, 39(5):1667–1713, 2010. 28

[HR18] G. H. Hardy and S. Ramanujan. Asymptotic formulae in combinatory analysis. *Proc. London Math. Soc.*, 17:75–115, 1918. 65

[HvtHJ$^+$11] Pinar Heggernes, Pim van 't Hof, Bart M. P. Jansen, Stefan Kratsch, and Yngve Villanger. Parameterized complexity of vertex deletion into perfect graph classes. In *FCT*, pages 240–251, 2011. 79

[HW12] Danny Hermelin and Xi Wu. Weak compositions and their applications to polynomial lower bounds for kernelization. In *SODA*, pages 104–113, 2012. 31

[Jan13] Bart M. P. Jansen. The power of data reduction: Kernels for fundamental graph problems. *Ph.D Thesis*, 2013. 167

[JLS14] Bart M. P. Jansen, Daniel Lokshtanov, and Saket Saurabh. A near-optimal planarization algorithm. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1802–1811. SIAM, 2014. 166

[JV12] Gwenaël Joret and Adrian Vetta. Reducing the rank of a matroid. *CoRR*, abs/1211.4853, 2012. 14, 207, 211

[KK15] Eun Jung Kim and O.-joung Kwon. A polynomial kernel for block graph deletion. In *10th International Symposium on Parameterized and Exact Computation, IPEC 2015, September 16-18, 2015, Patras, Greece*, volume 43 of *LIPIcs*, pages 270–281, 2015. 166

[Klo94] Ton Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994. 21

[KLP$^+$13] 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 International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, pages 613–624, 2013. 166

[KP14] Tomasz Kociumaka and Marcin Pilipczuk. Faster deterministic feedback vertex set. *Inf. Process. Lett.*, 114(10):556–560, 2014. 101

[KPPW12] Stefan Kratsch, Marcin Pilipczuk, Michal Pilipczuk, and Magnus Wahlström. Fixed-parameter tractability of multicut in directed acyclic graphs. In *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I*, pages 581–593, 2012. 206

[KPRR14] Stefan Kratsch, Marcin Pilipczuk, Ashutosh Rai, and Venkatesh Raman. Kernel lower bounds using co-nondeterminism: Finding induced hereditary subgraphs. *TOCT*, 7(1):4:1–4:18, 2014. 7

[KR02] Subhash Khot and Venkatesh Raman. Parameterized complexity of finding subgraphs with hereditary properties. *Theor. Comput. Sci.*, 289(2):997–1008, 2002. 4, 31, 32, 56, 237

[Kra12] Stefan Kratsch. Co-nondeterminism in compositions: a kernelization lower bound for a Ramsey-type problem. In *SODA*, pages 114–122, 2012. 4, 27, 28, 31, 39, 40, 43, 44, 48, 52

[KT11] Ken-ichi Kawarabayashi and Mikkel Thorup. The minimum k-way cut of bounded size is fixed-parameter tractable. In *Proceedings of the 52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 160–169. IEEE, 2011. 169, 206

[LNR$^+$14] 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. 9, 80, 168, 174, 180, 196, 203

[Lov83] L. Lovasz. Perfect graphs. In L. W. Beineke and R. J. Wilson, editors, *Selected Topics in Graph Theory, Volume 2*, pages 55–67. Academic Press, London-New York, 1983. 36, 57

[LR12] Daniel Lokshtanov and M. S. Ramanujan. Parameterized tractability of multiway cut with parity constraints. In *Automata, Languages, and Programming -*

39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I, pages 750–761, 2012. 188

[LY80] 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. 3, 32, 39, 79, 165

[LY94] Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41:960–981, September 1994. 79, 165

[Mar06] Dániel Marx. Parameterized graph separation problems. *Theoret. Comput. Sci.*, 351(3):394–406, 2006. 206

[Mar10] Dániel Marx. Chordal deletion is fixed-parameter tractable. *Algorithmica*, 57(4):747–768, 2010. 34, 79

[MOR13] 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. 165, 206, 207

[MPR$^+$13] Neeldhara Misra, Fahad Panolan, Ashutosh Rai, Venkatesh Raman, and Saket Saurabh. Parameterized algorithms for max colorable induced subgraph problem on perfect graphs. In *Graph-Theoretic Concepts in Computer Science - 39th International Workshop, WG 2013, Lübeck, Germany, June 19-21, 2013, Revised Papers*, pages 370–381, 2013. 8

[MR14] Dániel Marx and Igor Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset. *SIAM J. Comput.*, 43(2):355–388, 2014. 206

[MS12] Dániel Marx and Ildikó Schlotter. Obtaining a planar graph by vertex deletion. *Algorithmica*, 62(3-4):807–822, 2012. 34, 79

[NSS95] Moni Naor, Leonard J. Schulman, and Aravind Srinivasan. Splitters and near-optimal derandomization. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 182–191. IEEE, 1995. 58

[NSS01] Assaf Natanzon, Ron Shamir, and Roded Sharan. Complexity classification of some edge modification problems. *Discrete Applied Mathematics*, 113(1):109–128, 2001. 79

[Oxl10] James G. Oxley. *Matroid theory*, volume 21 of *Oxford Graduate Texts in Mathematics*. Oxford university press, 2nd edition, 2010. 119, 120

[PPW15] Marcin Pilipczuk, Michal Pilipczuk, and Marcin Wrochna. Edge bipartization faster than 2^k. *CoRR*, abs/1507.02168, 2015. 168, 177

[PRS15] Geevarghese Philip, Ashutosh Rai, and Saket Saurabh. Generalized pseudoforest deletion: Algorithms and uniform kernel. In *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part II*, volume 9235 of *Lecture Notes in Computer Science*, pages 517–528. Springer, 2015. 11

[RR16] Ashutosh Rai and M. S. Ramanujan. Strong parameterized deletion: Bipartite graphs. In *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016, December 13-15, 2016, Chennai, India*, pages 21:1–21:14, 2016. 13

[RRS16] Ashutosh Rai, M. S. Ramanujan, and Saket Saurabh. A parameterized algorithm for mixed-cut. In *LATIN 2016: Theoretical Informatics - 12th Latin American Symposium, Ensenada, Mexico, April 11-15, 2016, Proceedings*, pages 672–685, 2016. 15, 231

[RS04] Neil Robertson and Paul D. Seymour. Graph minors. xx. wagner's conjecture. *J. Combinatorial Theory Ser. B*, 92(2):325–357, 2004. 136

[RS08] Venkatesh Raman and Saket Saurabh. Short cycles make w-hard problems hard: Fpt algorithms for w-hard problems in graphs with no short cycles. *Algorithmica*, 52(2):203–225, 2008. 7, 56

[RS15] Ashutosh Rai and Saket Saurabh. Bivariate complexity analysis of almost forest deletion. In *Computing and Combinatorics - 21st International Confer-*

ence, COCOON 2015, Beijing, China, August 4-6, 2015, Proceedings, pages 133–144, 2015. 10

[RST94] Neil Robertson, Paul D. Seymour, and Robin Thomas. Quickly excluding a planar graph. *J. Comb. Theory, Ser. B*, 62(2):323–348, 1994. 127

[RSV04] Bruce A. Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Oper. Res. Lett.*, 32(4):299–301, 2004. 10, 101, 207

[TC90] R. I Tyshkevich and A. A. Chernyak. Yet another method of enumerating unmarked combinatorial objects. *Mathematical Notes*, 48:1239–1245, 1990. 79

[TIAS77] Shuji Tsukiyama, Mikio Ide, Hiromu Ariyoshi, and Isao Shirakawa. A new algorithm for generating all the maximal independent sets. *SIAM J. Comput.*, 6(3):505–517, 1977. 59

[vBKMN10] René van Bevern, Christian Komusiewicz, Hannes Moser, and Rolf Niedermeier. Measuring indifference: Unit interval vertex deletion. In *WG*, pages 232–243, 2010. 34

[vtHV13] Pim van 't Hof and Yngve Villanger. Proper interval vertex deletion. *Algorithmica*, 65(4):845–867, 2013. 34

[Woe01] Gerhard J. Woeginger. Exact algorithms for np-hard problems: A survey. In *Combinatorial Optimization - Eureka, You Shrink!, Papers Dedicated to Jack Edmonds, 5th International Workshop, Aussois, France, March 5-9, 2001, Revised Papers*, pages 185–208, 2001. 17

[Yan78] Mihalis Yannakakis. Node-and edge-deletion NP-complete problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, STOC '78, pages 253–264, New York, NY, USA, 1978. ACM. 165

[YG87] Mihalis Yannakakis and Fanica Gavril. The maximum k-colorable subgraph problem for chordal graphs. *Inf. Process. Lett.*, 24(2):133–137, January 1987. 7, 56, 74