The Parameterized Complexity of Graph Editing & MaxMin Problems

विद्या वाचस्पति की
उपाधि की अपेक्षाओं की आंशिक पूर्ति में प्रस्तुत शोध प्रबंध

A thesis submitted in partial fulfillment of the requirements of the degree of Doctor of Philosophy

द्वारा / By
हितेन्द्र कुमार / Hitendra Kumar

पंजीकरण सं. / Registration No.: 20172025

शोध प्रबंध पर्यवेक्षक / Thesis Supervisor:
प्रो.सौमेन मैती / Prof. Soumen Maity

भारतीय विज्ञान शिक्षा एवं अनुसंधान संस्थान पुणे

INDIAN INSTITUTE OF SCIENCE EDUCATION AND RESEARCH PUNE

2025

Certificate

Certified that the work incorporated in the thesis entitled "*The Parameterized Complexity of Graph Editing & MaxMin Problems* ", submitted by *Hitendra Kumar* was carried out by the candidate under my supervision. The work presented here or any part of it has not been included in any other thesis submitted previously for the award of any degree or diploma from any other university or institution.

Prof. Soumen Maity, IISER Pune

Committee:

Prof. Soumen Maity, IISER Pune

Prof. Saket Saurabh, IMSc Chennai

Dr. Vivek Mohan Mallick, IISER Pune

This thesis is dedicated to *mor* Chhattisgarh

Declaration

I declare that this written submission represents my ideas in my own words. Where others' ideas have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity. I have not misrepresented, fabricated, or falsified any idea, data, fact, or source in my submission. I understand that violation of the above will result in disciplinary action by the institute and may also lead to penal action if proper citation or permission has not been obtained from the sources used.

The work reported in this thesis is the original work done by me under the guidance of Prof. Soumen Maity.

Hitendra Kumar

Acknowledgments

I wish to begin by expressing my deepest gratitude to my advisor, Prof. Soumen Maity. His guidance, expertise, and unwavering support have been instrumental in shaping the course of my research. I am deeply thankful for the time and patience he offered as I ventured into a completely new field, and for allowing me the freedom to explore not just academia, but life beyond it. His mentorship has significantly contributed to my growth—not only as a researcher, but also as an individual.

I am sincerely grateful to Prof. Saket Saurabh, a member of my Research Advisory Committee, for his thoughtful insights and encouragement throughout my academic journey. I also thank Dr. Vivek Mallick for being a part of my committee and for his valuable suggestions that helped refine my work.

A special note of appreciation goes to Ajinkya, who has been an integral part of my journey—not just as a collaborator, but as an *unofficial co-supervisor* and a friend. I learned immensely from him, and I will always cherish his patience in answering even my silliest questions, and his clarity in dissecting complex problems.

I would also like to thank Roohani for the countless insightful discussions—discussions that continue to this day and constantly challenge and inspire me to think deeper. These conversations have been a significant part of my intellectual growth. I would like to thank Mihir, Ajaykrishnan, and Shuvam for the many insightful discussions during the initial phase of my PhD.

My sincere thanks go to the administrative staff, especially Mrs. Suvarna Bhardwaj, Mr. Yogesh Kolap, Mrs. Sayalee Damle, and Mr. Swapnil Bhuktar, for their consistent support. I am equally grateful to the housekeeping and security staff for maintaining such a welcoming

and vibrant environment that made the campus a wonderful place to work.

To my *friends*—Namrata, Tushar, Prachi, Raman, Clifford, Ronit, Kundan, Sumit, and Roop—thank you for all the laughter we shared, the lame jokes that lifted our spirits, and your constant presence during the difficult times. I have been fortunate to make many friends during this journey—too many to name—but to all of you, I extend my heartfelt thanks.

I would also like to thank the Aroha Club and Hindi Club for nurturing the artist in me, and the Disha Club for giving me the opportunity to serve as a coordinator for the Prerna Program. To all the students from Prerna—thank you for contributing to my personal growth in ways I never imagined.

A special thanks to all the BS-MS students, who unknowingly helped transform me from an introvert to an ambivert. And last but not least, my gratitude goes to every player on the Zeher team—thank you for being part of a sport-loving family that brought so much energy and joy into my life.

Abstract

This thesis presents a comprehensive study of the parameterized complexity of several graph-theoretic problems, with a focus on graph modification, separation, and vertex splitting. The investigation is primarily motivated by two thematic directions: enforcing structural properties in graphs via modification operations, and analyzing maximization problems that seek large minimal solutions under certain constraints (commonly referred to as MaxMin problems).

We begin with an in-depth analysis of *Uniform Cluster Graph Modification* problems, where the objective is to transform a given graph into a disjoint union of equal-sized cliques using a limited number of edit operations, including vertex deletion, edge deletion, edge addition, edge editing, and vertex splitting. For various problem variants, we provide improved fixed-parameter tractable (FPT) algorithms and kernelization results, including both polynomial and linear kernels. Our contributions resolve several open questions posed in the existing literature on these problems.

Subsequently, we examine a range of *MaxMin Problems*, such as *MaxMin Degree-d-Modulator* and *MaxMin Feedback Vertex Set*. We extend known results for special cases, establish new kernelization bounds, and design FPT algorithms under natural structural parameters, including vertex cover and treewidth. We also study generalizations of classical separation problems by introducing and analyzing the *MaxMin Multiway Cut-Uncut*, *MaxMin Separator-Unseparator*, and *MaxMin Subset Feedback Vertex Set* problems, presenting both algorithmic results and hardness proofs with respect to various parameterizations.

Finally, the thesis investigates the computational complexity of $\mathcal{F}$*-Vertex Splitting* for different graph classes under both inclusive and exclusive splitting models. We provide a fine-grained complexity classification, proving NP-hardness for several variants.

Overall, this work advances the theoretical understanding of graph modification, separation, and vertex splitting problems within the framework of parameterized complexity, offering novel algorithmic techniques, kernelization bounds, and hardness results for a broad spectrum of combinatorial problems.

Contents

Chapter 1

Introduction

The foundational result establishing the computational intractability of many problems was presented by Cook in 1971, when the Boolean Satisfiability Problem (SAT) was proven to be NP-complete [35]. Building on this, in 1972, Karp [82] identified 21 fundamental problems as NP-complete, marking the beginning of a vast landscape of computationally difficult problems. Since then, a multitude of problems have been shown to be NP-hard. Assuming the widely believed conjecture P $\neq$ NP, these problems are not expected to admit polynomial-time algorithms in terms of the input size.

To cope with the inherent difficulty of NP-hard problems, several algorithmic frameworks have been developed, including approximation algorithms, randomized algorithms, and exact exponential-time algorithms. These approaches often trade off between solution quality and computational efficiency.

In the early 1990s, Downey and Fellows introduced the theory of *Parameterized Complexity*, offering a refined lens to analyze computational problems. This framework evaluates the complexity of a problem not only with respect to the input size but also in terms of an auxiliary measure called the *parameter*. The most natural choice for the parameter is often the solution. However, in many contexts, structural features of the input—such as *treewidth*, *vertex cover number*, or *feedback vertex set size*—serve as effective parameters.

The central objective in parameterized complexity is to discover meaningful parameterizations under which computationally hard problems can be solved efficiently. Specifically, a

problem is said to be *fixed-parameter tractable (FPT)* if it can be solved in time $f(k) \cdot n^{\mathcal{O}(1)}$, where n is the input size, k is the parameter, and f is a computable function independent of n.

Not all parameterized problems admit FPT algorithms. To classify their relative difficulty, a hierarchy of complexity classes known as the *W-hierarchy* has been developed, analogous to the role of NP-completeness in classical complexity theory. Demonstrating that a parameterized problem is $\mathsf{W}[t]$-hard for some $t \geq 1$ serves as strong evidence that it is unlikely to be fixed-parameter tractable.

1.1 Graph Modifications

Graph modification problems, where the goal is to modify a given graph to meet specific desired properties, have gained significant attention in recent decades. These types of problems have become a rich source of inspiration for developing new techniques in the fields of parameterized algorithms and computational complexity. Researchers have focused on exploring how various modifications, such as adding or removing edges or vertices, can transform a graph to satisfy certain structural or functional criteria. This has led to important advancements in both theoretical frameworks and practical algorithms, driving forward our understanding of graph theory and its applications in computer science.

For a family of graphs $\mathcal{F}$, the $\mathcal{F}$-Editing takes as an input a graph G and an integer k, and the objective is to decide if at most k *edit operations* on G can result in a graph that belongs to $\mathcal{F}$. Some key graph editing operations include deleting vertices, deleting edges, adding edges, contracting edges, and splitting vertices. The majority of studies on $\mathcal{F}$-Editing have focused on combinations of vertex deletion, edge deletion, edge addition and edge contraction. It is only recently that vertex splitting as an edit operation has begun to attract attention. A *vertex split* is a graph modification where a vertex v is replaced by two new vertices, v_1 and v_2. Each edge that was originally incident to v is then reassigned to v_1 or v_2. We have some variants of vertex splitting that we will define later.

When the operations are limited to only vertex/edge deletion, the corresponding problem is referred to as the $\mathcal{F}$-Vertex/Edge Deletion. If we only allow edge addition, then the corresponding problem is called $\mathcal{F}$-Edge Addition. If we allow edge addition as well as

edge deletion, then the corresponding problem is called $\mathcal{F}$-Edge Editing. If we only allow edge contraction, then the corresponding problem is called $\mathcal{F}$-Edge Contraction. If we only allow vertex splitting, then the corresponding problem is called $\mathcal{F}$-Vertex Splitting.

A foundational result in graph modification problems is the work by Lewis and Yannakakis [88], which establishes the necessary and sufficient conditions for vertex-deletion problems to be solvable in polynomial time, assuming $\mathsf{P} \neq \mathsf{NP}$, for hereditary properties. However, no similar dichotomy has been established for edge-removal problems. Many parameterized and kernelization algorithms for vertex deletion problems, including Vertex Cover, Feedback Vertex Set, Odd Cycle Transversal and many others can be found in the books by Cygan et al. [40] and Fomin et al. [62]. We refer to [27, 98], for an overview of results regarding the complexity of $\mathcal{F}$-Edge Deletion. For $\mathcal{F}$-Edge Modification, we refer to a survey [39], which also has some open problems. For the *hereditary graph* class, defined by a finite set of *forbidden induced subgraphs*, a simple *branching algorithm* shows that $\mathcal{F}$-Vertex/Edge Deletion, $\mathcal{F}$-Edge Addition as well as $\mathcal{F}$-Edge Editing are all fixed-parameter tractable [29]. The central idea in getting FPT-algorithms was to destroy a structure which forbids the input graph from being in $\mathcal{F}$. $\mathcal{F}$-Edge Contraction is NP-Hard when $\mathcal{F}$ is family of *planar graphs, outerplanar graphs, series-parallel graphs, forests* and *chordal graphs*. We refer to [107] for an overview of results regarding the complexity of $\mathcal{F}$-Edge Contraction.

This thesis focuses on the parameterized complexity and algorithmic aspects of a collection of important graph modification problems. The problems studied include: *Uniform Cluster Graph Modifications*, *Maximum Minimal Cut-Uncut*,*Maximum Minimal Separator-Unseparator*, *Maximum Minimal List Allocation*, *Maximum Minimal Feedback Vertex Set*, *Maximum Minimal Degree-d-Modulator*, and $\mathcal{F}$-Vertex Splitting. In the subsequent sections, we provide a detailed, section-wise treatment of each of these problems.

1.2 Uniform Cluster Graph Modification

Many real-world problems require grouping objects or data points into well-defined clusters. These problems can be modeled using graphs, where nodes represent objects and edges represent relationships. Graph modification techniques help enforce structural properties, such as equal-sized or highly similar clusters. Editing a graph—by adding or removing

edges, or deleting vertices—can make clustering more accurate or computationally feasible, especially when the original data is not well-structured.

A classic problem in this domain is the Cluster Vertex Deletion, where the objective is to delete at most k vertices from a graph such that the remaining graph becomes a *cluster graph*. A cluster graph is defined as a graph in which every connected component is a clique (i.e., a complete graph). The parameterized version of this problem has been the subject of extensive study, leading to the development of the best-known FPT algorithm with a running time of $\mathcal{O}^*(1.811^k)$ [1][108], and a *vertex kernel* of size $\mathcal{O}(k^2)$ [2]. Earlier contributions to this line of work have explored various facets of the problem, including complexity analysis and algorithmic strategies [23, 29, 71, 79].

Another well-studied problem in graph clustering is the Cluster Edge Deletion. The objective of the problem is to determine whether a given graph can be transformed into a cluster graph by deleting at most k edges. The parameterized complexity of Cluster Edge Deletion has received considerable attention in the literature [20, 30, 31, 41, 56, 71, 72, 73, 109]. The problem is solvable in $\mathcal{O}^*(1.404^k)$ time [109], and it admits a kernel with at most $2k$ vertices [30, 31].

Another notable problem in clustering is the Cluster Edge Editing, also known as Correlation Clustering. In this problem, the goal is to decide whether a graph can be transformed into a cluster graph by adding or deleting at most k edges. The problem was first introduced by Ben-Dor, Shamir and Yakhini [13], who were motivated by challenges in computational biology and independently by Bansal, Blum and Chawla, who applied the problem to document clustering in machine learning. Various parameterized versions of Cluster Editing and its many variants have since been intensively studied in the literature [18, 19, 20, 21, 28, 42, 54, 72, 74, 75, 86, 102].

In this thesis, we explore a relatively recent variant of the clustering problem. Our focus is on a computational problem where the goal is to modify a given graph—through vertex deletions, vertex splittings, edge deletions, additions, or edits—so that it becomes a collection of equal-sized cliques, referred to as *Uniform Cluster graphs*. Specifically, we study the Uniform Cluster Vertex Deletion (UCVD) problem, which asks whether at most k vertices can be deleted to achieve this structure. We also examine edge-based variants: Uniform Cluster Edge Deletion (UCED), Uniform Cluster Edge Addition

[1]The $\mathcal{O}^*(f(k))$ notation hides polynomial factors depending on n, where n is the input size.

(UCEA), and Uniform Cluster Edge Editing (UCEE), where the aim is to apply a limited number of respective operations to obtain a Uniform Cluster graph.

Recently, Firbas et al. [59] studied a problem called Cluster Vertex Splitting (Cluster-VS). The task in this problem is to transform an input graph into a cluster graph by performing at most k vertex splitting operations. A vertex split is a graph operation where a vertex v is replaced by two copies, say v_1 and v_2, with the combined neighborhood of the two copies being equal to the original neighborhood of v. Abu- Khzam et al. [3] propose two different vertex splitting operations: one (exclusive splitting) where v_1 and v_2 are required to have disjoint neighborhoods and another (inclusive splitting) where they are allowed to share neighbors (See Figure 1.1). Intuitively, given a graph G, the goal is to find cliques that cover all the edges of the graph with limited overlap between them. It has been shown that Cluster-VS admits a kernel with at most $3k + 3$ vertices. Similarly, we consider the Uniform Cluster Vertex Splitting (UCVS), where the task is to decide whether there is a sequence of at most k vertex splits to get a uniform cluster graph.

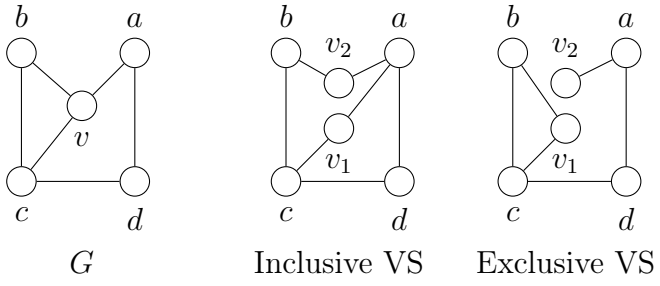


Figure 1.1: Inclusive and Exclusive vertex splitting of vertex v

Now, we formally define some problems that we consider in this thesis.

Uniform Cluster Vertex Deletion (UCVD)
Input: An undirected graph $G = (V, E)$ and a positive integer k.
Question: Is there a subset $S \subseteq V$ with $\|S\| \leq k$ such that $G - S$ is a collection of equal-sized cliques?

Uniform Cluster Edge Deletion (UCED)
Input: An undirected graph $G = (V, E)$ and a positive integer k. **Question:** Is there a subset $F \subseteq E$ with $\|F\| \leq k$ such that $G - F = (V, E \setminus F)$ is a collection of equal-sized cliques?

Uniform Cluster Edge Addition (UCEA)
Input: An undirected graph $G = (V, E)$ and a positive integer k. **Question:** Is there a subset $F \subseteq \binom{V(G)}{2}$ with $\|F\| \leq k$ such that $G + F = (V, E \cup F)$ is a collection of equal-sized cliques?

Uniform Cluster Edge Editing (UCEE)
Input: An undirected graph $G = (V, E)$ and a positive integer k. **Question:** Is there a subset $F \subseteq \binom{V(G)}{2}$ with $\|F\| \leq k$ such that $G \triangle F = (V, E \Delta F)$ is a collection of equal-sized cliques?

Uniform Cluster Vertex Splitting (UCVS)
Input: An undirected graph $G = (V, E)$ and a positive integer k. **Question:** Is there a sequence of at most k vertex splits that transforms G into a collection of equal-sized cliques?

Our contributions span kernelization, fixed-parameter tractability (FPT) algorithms, and structural parameterizations for above mentioned problems. For UCVD, we present a vertex kernel of size $\mathcal{O}(k^3)$ and an FPT algorithm with a running time of $\mathcal{O}^*(2^k)$, both improving upon prior work. We show that UCIVS and UCEVS are NP-hard and provide linear vertex kernels of size $4k$ for each. In the edge modification category, we obtain a vertex kernel of size $\mathcal{O}(k^2)$ for UCEE. For UCED, we provide a vertex kernel of size $6k$ and an FPT algorithm with a running time of $\mathcal{O}^*(1.47^k)$. For UCEA, we provide a linear vertex kernel of size $5k$. We also explore parameterizations based on structural measures of the input graph.

For a fixed positive integer r and a given budget k, the r-Eigenvalue Vertex Deletion (r-EVD) asks whether there exists a set $S \subseteq V(G)$ with at most k vertices such that the adjacency matrix of the graph $G \setminus S$ has at most r distinct eigenvalues. Variants of this problem, including edge deletion, edge addition, and edge editing, are defined analogously. It is known that the adjacency matrix A_G of a graph G has at most two distinct

eigenvalues if and only if G is a disjoint union of cliques of equal size. Consequently, the 2-Eigenvalue Vertex Deletion (2-EVD) can be reformulated as finding a set $S \subseteq V(G)$ of vertices such that the graph $G \setminus S$ becomes a disjoint union of cliques, each of size ℓ, where $1 \leq \ell \leq |V(G)|$. In other words, Uniform Cluster Vertex Deletion (UCVD) and 2-Eigenvalue Vertex Deletion (2-EVD) are equivalent problems for a graph. The same can be concluded for other editing operations.

1.3 Maximum Minimal Problems

Optimization problems have a well-defined structure: a set of problem instances, denoted by $\mathcal{I}$, and for a given instance $x \in \mathcal{I}$, a set of feasible solutions, each feasible solution is associated with a measure, and the objective is to either maximize or minimize the measure. The goal of an optimization problem is to find a feasible solution whose measure is either minimum or maximum among all possible feasible solutions. When the goal is to maximize the measure, the problem is called a *maximization* problem, and when the goal is to minimize the measure, it is referred to as a *minimization* problem. This thesis is concerned with classes of problems called *MaxMin* versions of problems. In the decision version of a minimization (resp. maximization) problem, usually the goal is to find a set of size at most k (resp. at least k) satisfying given constraints. In the MaxMin (resp. MinMax) version of the same minimization (resp. maximization) problem, we are interested in finding a minimal (resp. maximal) set of size at least k (resp. at most k) which satisfies the same set of constraints.

Over recent years, MaxMin and MinMax problems have been found to possess an interesting combinatorial structure of their own and have now become an object of more widespread study from the perspective of Approximation Algorithms and Parameterized Complexity. The initial motivation for studying these variants stems from the fact that the solution size in the MaxMin versions provides insight into the worst-case performance of greedy heuristics. Studies have shown that the MaxMin and MinMax variants present significant challenges and are highly resistant to most established methods for addressing the classical version of the problem. MaxMin or MinMax versions of Vertex Cover [24, 114], Dominating Set [1], Edge Dominating Set [69, 93, 96], Cut and Separations [50, 81, 76], Knapsack [63], Matching [33], and Coloring [12] have been studied.

In the realm of parameterized complexity, Zehavi [114] and Boria et al. [24] studied

MaxMin Vertex Cover. We know that a set of vertices $X \subseteq V(G)$ of a graph G is vertex cover if and only if its complement $V(G) \setminus X$ is an independent set. In other words, if X is a vertex cover then $G-X$ has degree 0. Now, we define the Bounded-Degree Vertex Deletion (BDD), where the goal is to identify whether it is possible to remove at most k vertices from an undirected graph G such that the resulting graph has a maximum degree of at most d where d is a specified degree bound. Note that for $d = 0$, BDD is the same as Vertex Cover. In addition to being a natural generalization of the classical Vertex Cover problem, the Bounded-Degree Vertex Deletion (BDD) has practical applications in fields such as computational biology and social network analysis. The parameterized complexity of BDD has already been studied [15, 38, 55, 97, 99] extensively. In this thesis, we will fix d and we will address the problem as Degree-d-Modulator.

Definition 1. *A vertex subset $S \subseteq V$ is called a* degree-d-modulator *if $G - S$ has maximum degree at most d.*

Definition 2. *A degree-d-modulator S is called a* minimal degree-d-modulator *if for any $v \in S$, $S \setminus \{v\}$ is not a degree-d-modulator. In other words, $G - (S \setminus \{v\})$ has a vertex of degree at least $d + 1$.*

Figure 1.2 shows an example of a minimal degree-2-modulator.

Original graph G. Graph $G - S$ (maximum degree = 2).

Figure 1.2: A graph G where $S = \{x, y\}$ is a minimal degree 2-modulator

We can easily observe that if S is minimal degree-d-modulator, then every $v \in S$ either has at least $d + 1$ neighbours in $G - S$ or is adjacent to a vertex $u \in V \setminus S$, which has exactly d neighbours in $G - S$. Now, we will define our problem formally.

MaxMin Degree-d-Modulator
Input: An undirected graph $G = (V, E)$ and an integer $k \in \mathbb{N}$.
Question: Does G admit a minimal degree-d-modulator of size greater than or equal to k?

In this thesis, we undertake a comprehensive study of the parameterized complexity of the MaxMin Degree-d-Modulator problem for arbitrary values of d. We begin by demonstrating that the problem admits a polynomial vertex kernel of size $\mathcal{O}(dk^2 + d^2k)$, thereby extending the known kernelization result for the special case $d = 0$. Furthermore, we establish that MaxMin Degree-d-Modulator is fixed-parameter tractable when parameterized by the *treewidth* of the input graph.

Feedback Vertex Set (FVS) together with Vertex Cover are arguably the two most well studied problems in parameterized complexity. In FVS, we are given an undirected graph G, a positive integer k, and the question is to check whether G has a vertex set S of size at most k such that $G - S$ is a forest (an acyclic graph). The set S is called *feedback vertex set* or *fvs* in short. Several minimization variants of FVS have been studied in the literature such as finding a set S such that $G - S$ is acyclic and $G[S]$ is connected or $G[S]$ is an independent set. In this thesis, we consider a maximization version of FVS, namely MaxMin FVS.

Definition 3. *A set S is called a* minimal fvs*, if S is an fvs and for every $v \in S$, $S \setminus \{v\}$ is not an fvs. That is, no proper subset of S is an fvs.*

Figure 1.3 shows an example of a minimal feedback vertex set.

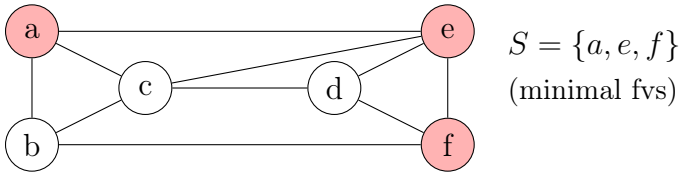


Figure 1.3: The set $S = \{a, e, f\}$ is a feedback vertex set. It is *minimal* because no proper subset of S is a feedback vertex set.

It is not hard to see that if S is a minimal FVS, then every $u \in S$ has a *private cycle*, that is, there exists a cycle in $G[(V(G) \setminus S) \cup \{u\}]$, which goes through u. It is not surprising that MaxMin FVS is known to be NP-complete and is considerably more challenging than Minimum FVS in various aspects, including its approximability and its suitability for algorithms addressing special cases. Now we are ready to define the problem formally.

MaxMin Feedback Vertex Set (MMFVS)
Input: An undirected graph $G = (V, E)$ and an integer $k \in \mathbb{N}$.
Question: Does G admit a minimal FVS of size greater than or equal to k?

Upper Dominating Set (which is the standard name for MaxMin Dominating Set) and MaxMin VC are well-studied problems, both in the context of approximation and in the context of parameterized complexity [1, 11, 24, 25, 34, 37, 77, 81, 114, 115]. MaxMin FVS seems to lie in an intermediate space between two more well-studied problems of this type: MaxMin Vertex Cover, for which the best achievable approximation ratio is $\sqrt{n}$, and Upper Dominating Set, which does not admit any $n^{1-\epsilon}$ approximation. Dublois et al. [48] have confirmed and quantified this intuition by showing the first non-trivial polynomial time approximation for MaxMin FVS with a ratio of $O(n^{\frac{2}{3}})$, as well as a matching hardness of approximation bound of $n^{\frac{2}{3}-\epsilon}$, improving the previous known [94] (shown by Mishra and Sikdar) hardness of $n^{\frac{1}{2}-\epsilon}$.

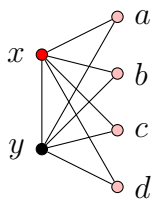


Figure 1.4: Graph G with $\mathsf{fvs}(G) = 1$ and $\mathsf{opt}(G) = 4$

Let $\mathsf{fvs}(G)$ denote the minimum size of a feedback vertex set of G, and let $\mathsf{opt}(G)$ denote the maximum size of a minimal feedback vertex set of G. Clearly, $\mathsf{fvs}(G) \leq \mathsf{opt}(G)$. The gap between these quantities can be arbitrarily large – as shown in Figure 1.4. Here, $\mathsf{fvs}(G) = 1$, while $\mathsf{opt}(G) = |V(G)| - 2$. Further, observe that the same example also shows that the gap between $\mathsf{vc}(G)$ and $\mathsf{opt}(G)$ can be arbitrarily large. Here, $\mathsf{vc}(G) = 2$, while $\mathsf{opt}(G) = |V(G)| - 2$. The discussion above implies that for MaxMin FVS both $\mathsf{fvs}(G)$ and $\mathsf{vc}(G)$ are interesting parameters to consider.

We study the MaxMin Feedback Vertex Set (MMFVS) problem parameterized by vertex cover and present several new results. We design an FPT algorithm that runs in time $\mathcal{O}^*(2^{\mathcal{O}(\mathsf{vc}(G)\log \mathsf{vc}(G))})$. We complement this by proving that MMFVS does not admit a polynomial compression under this parameter unless coNP $\subseteq$ NP/poly. Additionally, we develop an FPT-approximation algorithm: for any fixed $\epsilon > 0$, there exists an algorithm with running time $2^{\mathcal{O}(\mathsf{vc}(G)/\epsilon)} \cdot n^{\mathcal{O}(1)}$ that outputs a minimal feedback vertex set of size at least $(1-\epsilon) \cdot \mathrm{opt}(G)$.

One must note that for understanding the parameterized complexity of MaxMin versions of problems such as Vertex Cover, Feedback Vertex Set and st-cut does not require

any new algorithmic ideas. Though constructing faster and more efficient algorithms does require non-trivial effort compared to their standard minimization counterpart. The standard method of treewidth based win-win approach provide an easy way to identify these problems are in the FPT class. A particularly intriguing problem in this domain is the MaxMin *st*-Separator, which was posed as an open problem by Hanaka, Bodlaender, van der Zanden and Oto [76].

Recently Gaikwad, Kumar, Maity, Saurabh and Sharma [66] resolved this open problem of Hanaka et al. in positive. This work is the first to leverage the concept of highly unbreakable instances (instances that cannot be separated into two big parts by using a small set of vertices) to establish the fixed-parameter tractability of MaxMin (separation) problems.

In this thesis, we generalize the results of Gaikwad et al. [66] by studying MaxMin versions of several NP-hard generalizations of *st*-Cut and *st*-Separator. One of the most interesting such generalizations is the MaxMin version of Multiway Cut-Uncut. In the MaxMin Multiway Cut-Uncut, given a graph G, a set of terminals $T \subseteq V(G)$, a partition $(T_1, \ldots, T_p)$ of the terminals T and a positive integer k, the goal is to find if there exists a set Z of at least k edges whose deletion results in a graph where, for each $i \in \{1, \ldots, p\}$, T_i is contained is some connected component *(uncut constraint)* and for each $i, j \in \{1, \ldots, p\}$, $i \neq j$, no vertex of T_i has a path to any vertex of T_j *(cut constraints)* and the set Z is inclusion-wise minimal with this property. Note that when for each $i \in \{1, \ldots, p\}$, $|T_i| = 1$, this corresponds to the MaxMin version of the classic Multiway Cut, which corresponds to *st*-Cut in the special case of $p = 2$.

The previous paragraph focused on generalization(s) of MaxMin *st*-Cut, that is on the edge-deletion variant of disconnecting s from t. Gaikwad et al. [66] showed that the MaxMin version of the vertex-deletion variant of disconnecting s and t in a graph is FPT. So we consider the vertex-deletion generalization, which we call MaxMin Multiway Separator-Unseparator, to avoid confusion. For this problem, we show that the problem soon becomes W[1]-hard and the only FPT case occurs for the special case of MaxMin Multiway Separator, which is the problem where given an undirected graph G, a set of terminals $T \subseteq V(G)$ and a positive integer k, determine if there exists a set $X \subseteq V(G) \setminus T$ such that $G - X$ has no path between t_i and t_j for any $t_i, t_j \in T$, $t_i \neq t_j$, and X is inclusion-wise minimal with respect to this property.

To show that MaxMin Multiway Separator is FPT, we show that a more general

problem MaxMin Subset Feedback Vertex Set is FPT. MaxMin Subset Feedback Vertex Set generalizes two existing problems in the literature: MaxMin Feedback Vertex Set studied by Lampis, Nikolaos and Manolis [87] and MaxMin st-Separator studied by Gaikwad et al. [66].

Now we formally define some of the problems considered in this thesis.

MaxMin Multiway Cut-Uncut
Input: An undirected graph G, a set $T \subseteq V(G)$ of terminals together with a partition $(T_1, \ldots, T_p)$ of T, and a positive integer k.
Question: Does there exist a set $X \subseteq E(G)$ of size at least k, such that X is minimal and in $G - X$, each T_i is entirely contained in a unique connected component?

MaxMin Separator-Unseparator
Input: An undirected graph G, a set $T \subseteq V(G)$ of terminals together with a partition $(T_1, \ldots, T_p)$ of T, and a positive integer k.
Question: Does there exist a set $S \subseteq V(G) \setminus T$ of size at least k, such that S is minimal and in $G - S$, each T_i is contained in a unique connected component?

MaxMin Subset Feedback Vertex Set
Input: An undirected graph G, a set $T \subseteq V(G)$ and a positive integer k.
Question: Does there exist $X \subseteq V(G)$ of size at least k, such that X is a minimal set with the property that in $G - X$ there is no cycle that intersects the set T?

Figure 1.3 shows an example of a minimal subset feedback vertex set.

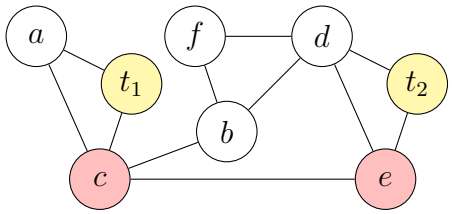


Figure 1.5: A connected graph G with terminal set $T = \{t_1, t_2\}$ (yellow). The set $X = \{c, e\}$ (red) is a minimal subset feedback vertex set of size 2.

Finally, we consider another generalization of the Multiway Cut called the List Allocation, which was introduced by Kim, Paul, Sau, and Thilikos [85]. In the classical List

Allocation, given a graph G, a positive integer r, a set of non-negative integers α_{ij} for $1 \leq i < j \leq r$, and a list function for the vertices $\lambda : V(G) \to 2^{[r]}$, the goal is to determine if there is a partition of vertex set into r parts $V_1, V_2, \ldots, V_r$ such that $|E(V_i, V_j)| = \alpha_{ij}$ and $v \in V_l$ implies $l \in \lambda(v)$. Kim et al. [85] showed that List Allocation is FPT when parameterized by $k = \sum_{1 \leq i < j \leq r} \alpha_{ij}$. Chandrasekaran and Wang [32] studied a variant (with relaxed input that $\lambda(v) = [r]$) of this problem where the goal is to bound the maximum number of edges going across two parts by α. This problem turned out to be W[1]-hard when parameterized by α. The authors provided an interesting FPT approximation algorithm.

In the MaxMin List Allocation (the one that we consider in this work), the goal is opposite to that of Chandrasekaran and Wang [32], that is, one needs to determine if there exists a partition of vertices into r non-empty parts such that the number of edges across any two parts satisfies the given lower bound. Formally, In the MaxMin List Allocation, given a graph G, a positive integer r, a set of non-negative integers α_{ij} for $1 \leq i < j \leq r$, and a list function for the vertices $\lambda : V(G) \to 2^{[r]}$, the goal is to partition $V(G)$ into r non-empty parts $V_1, \ldots, V_r$ such that $|E(V_i, V_j)| \geq \alpha_{ij}$ for all $1 \leq i < j \leq r$, and every vertex $v \in V_l$ satisfies $l \in \lambda(v)$.

Here we formally define the problem studied in this thesis.

Input: An undirected graph G, a function $\lambda : V(G) \to 2^{[r]}$, a set of non-negative integers α_{ij} and a positive integer r.
Question: Does there exists a partition of $V(G)$ into r non-empty parts $V_1, V_2, \ldots, V_r$ such that $|E(V_i, V_j)| \geq \alpha_{ij}$ for all $1 \leq i < j \leq r$ and $v \in \lambda(v)$ for all $v \in V(G)$?

Note that MaxMin List Allocation generalizes the MaxMin Multiway Cut-Uncut if we enforce the condition that $G[V_i]$ is connected for all $i \in \{r\}$. We consider two parameters $\alpha_{\max} = \max_{1 \leq i < j \leq r} \alpha_{ij}$ and r.

An application of color coding allows us to construct an FPT algorithm when parameterized by $r + \alpha_{\max}$. The problem is NP-hard even when $r = 2$. Therefore, the interesting question remains whether the problem is FPT when parameterized by $\alpha_{\max}$ only. Unfortunately, this is not the case. We show that the problem turns out to be W[1]-hard by providing an FPT reduction from the Biclique problem.

1.4 Vertex Splitting Problems

A *vertex split* is a graph modification that replaces a vertex v with two new vertices, v_1 and v_2, and assigns each edge previously incident to v to either v_1, v_2, or both. Through a sequence of vertex splits, a given graph can be systematically transformed to achieve a specific desired property. This process gives rise to a distinct combinatorial optimization problem and an associated decision problem for each possible graph property. In this thesis, we explore the classical computational complexity of this family of problems and also conclude some results about their parameterized complexity. Vertex splitting was primarily used in the context of graph planarization, but it is now used in various settings.

Vertex splitting has been applied to the classical cluster editing problem. Now, we will take an interesting example from [9] to motivate how vertex splitting could be useful in real-life problems. In real-world networks, clusters frequently overlap. For instance, in social networks, an individual may be part of multiple communities, such as those related to work, school, or their neighborhood. Another compelling motivation arises in language networks, where clustering words based on similar usage can be challenging due to homonyms—words with multiple meanings, like "bat". Vertex splitting is one natural approach to incorporating such overlap into cluster editing. There are three distinct variants of vertex splitting that have been proposed in the literature. Each variant is suited to different types of problems or scenarios, and the choice of which variant to use depends on the specific context or requirements of the situation at hand. By carefully considering the characteristics of the problem, one can select the most appropriate variant to achieve the desired outcome.

For each fixed graph class $\mathcal{F}$, we obtain a separate vertex splitting problem. In this thesis, we study the classical complexity of $\mathcal{F}$-Vertex Splitting and get some parameterized complexity results as a consequence for some class $\mathcal{F}$.

$\mathcal{F}$-Vertex Splitting ($\mathcal{F}$-VS)
Input: An undirected graph $G = (V, E)$ and a positive integer k.
Question: Is there a sequence of at most k vertex splits that transforms G into a graph G' such that G' belongs to $\mathcal{F}$?

This thesis considers several different graph classes. We will begin by providing some definitions and then listing some of the problems addressed in this thesis.

Definition 4. *A graph G is called bipartite if its vertex set can be partitioned into two subsets V_1 and V_2, such that every edge connects a vertex from V_1 to a vertex from V_2.*

Definition 5. *A complete bipartite graph, or biclique, is a special kind of bipartite graph in which every vertex of the first set (say V_1) is connected to every vertex of the second set (say V_2).*

We say $p \times q$ is the dimension of the complete bipartite graph if $|V_1| = p$ and $|V_2| = q$. Note that $p \times q$ and $q \times p$ are considered the same dimension.

Figure 1.6 shows an example of a complete bipartite graph.

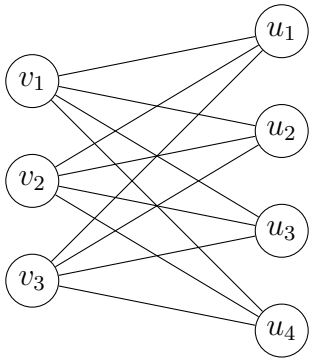


Figure 1.6: The complete bipartite graph $K_{3,4}$ has dimension 3×4 (or 4×3). Each vertex in the left partition (V_1) is adjacent to every vertex in the right partition (V_2).

Definition 6. *A uniform bicluster graph is a graph where each connected component is a complete bipartite graph of the same dimension.*

Definition 7. *A star graph is a complete bipartite graph $K_{1,n}$, where one part has a single vertex (the central vertex) and the other part contains n vertices (the leaves).*

Definition 8. *A star forest is a disjoint union of stars.*

Definition 9. *A cycle graph is a disjoint union of cycles.*

Now, we are ready to define our problems formally. One can easily observe that the problems we define correspond to $\mathcal{F}$-Vertex Splitting problem for some graph class $\mathcal{F}$.

Bipartite-Vertex Splitting (Bipartite-VS)
Input: An undirected graph $G = (V, E)$ and a positive integer k.
Question: Is there a sequence of at most k vertex splits that transforms G into a bipartite graph?

Uniform Bicluster-Vertex Splitting (Uniform Bicluster-VS)
Input: An undirected graph $G = (V, E)$ and a positive integer k.
Question: Is there a sequence of at most k vertex splits that transforms G into G' such that G' is a bicluster graph.

Start Forest-Vertex Splitting (Star Forest-VS)
Input: An undirected graph $G = (V, E)$ and a positive integer k.
Question: Is there a sequence of at most k vertex splits that transforms G into a star forest?

Cycle Graph-Vertex Splitting (Cycle Graph-VS)
Input: An undirected graph $G = (V, E)$ and a positive integer k.
Question: Is there a sequence of at most k vertex splits that transforms G into a cycle graph?

Uniform Cycle Graph-Vertex Splitting (Uniform Cycle Graph-VS)
Input: An undirected graph $G = (V, E)$ and a positive integer k.
Question: Is there a sequence of at most k vertex splits that transforms G into a disjoint union of equal-sized cycles?

We show that $\mathcal{F}$-Vertex Splitting is NP-complete for both inclusive and exclusive variants when $\mathcal{F}$ is the class of uniform cycle graphs, star forests, and uniform biclusters. For bipartite graphs, both variants are equivalent to Bipartite Vertex Deletion, inheriting its known FPT and kernelization results. We show that Cycle Graph-Exclusive Vertex Splitting is solvable in polynomial time.

1.5 Literature Survey and Our Contributions

In this section, we review the existing literature on the specific problems studied in this dissertation.

Uniform Cluster Graph Modifications: Misra et al. [95] examined Uniform Cluster Graph Modification problem in a different context, investigating the complexity of reducing

the number of distinct eigenvalues of an input graph by removing vertices or editing edges. A special case where the number of eigenvalues is reduced to at most two is equivalent to transforming a graph into a Uniform Cluster graph, as shown in previous works [43, 70]. It is proved that an adjacency matrix of a graph has at most two distinct eigenvalues if and only if the graph is a disjoint union of equal-sized cliques. Misra et al. [95] provided various results for this special case.

Furthermore, Madathil and Meeks [92] recently explored a generalization of this problem termed Balanced Cluster Editing, where, given a graph G, and two integers $0 \leq \eta \leq n$ and k, we are permitted to edit up to k edges. This editing should yield a cluster graph where the size difference between any two connected components does not exceed η. Notably, when $\eta = 0$, this problem reduces to the Uniform Cluster Editing. Their paper presented polynomial kernels for Balanced Cluster Completion, Balanced Cluster Deletion, and Balanced Cluster Editing. While they provided polynomial kernels for the general cases, we have achieved better bounds in all considered variants when $\eta = 0$. Our results were obtained simultaneously and independently from those by Madathil and Meeks.

Before the work of Madathil and Meeks [92], a closely related variant of Balanced Cluster Edge Modification, termed Difference-δ Balanced Clustering was studied in the master thesis of Steinvik [106]. In this variant, δ plays the same role as η in Balanced Cluster Edge Modification. In addition to edge modification, vertex deletion was also considered in the thesis. Steinvik also introduced a generalization of the Uniform Cluster Graph Modification, called Factor-α Balanced Clustering. In this problem, given a graph G and integers α and k, the goal is to perform at most k edit operations to obtain a cluster graph in which, for every pair of components C_i and C_j, the inequality $\alpha \cdot |V(C_i)| \geq |V(C_j)|$ holds. When $\alpha = 1$, this problem coincides with Uniform Cluster Graph Modification. In his thesis, Steinvik provided a vertex kernel of size $\mathcal{O}(k^4)$ for both Difference-δ Balanced Cluster Vertex Deletion and Factor-α Balanced Cluster Vertex Deletion. We obtain an improved kernel for the special case when $\delta = 0$ or $\alpha = 1$.

We prove that UCVD admits a kernel with $\mathcal{O}(k^3)$ vertices. We present an FPT algorithm that runs in time $\mathcal{O}^*(2^k)$, improving upon the FPT algorithms given in [95, 106], which have a running time of $\mathcal{O}^*(3^k)$. In this work, we consider two kinds of vertex split-

ting operations called inclusive and exclusive vertex splitting. We show that the Uniform Cluster-Inclusive Vertex Splitting and Uniform Cluster-Exclusive Vertex Splitting are NP-hard. In the realm of parameterized complexity, we provide a linear vertex kernel of size $4k$ for Uniform Cluster Exclusive Vertex Splitting and a linear vertex kernel of size $4k$ for Uniform Cluster Inclusive Vertex Splitting.

For the problem UCEE, we provide a kernel with $\mathcal{O}(k^2)$ vertices. Furthermore, we present linear kernels with $6k$ and $5k$ vertices for UCED and UCEA, respectively, improving upon the quadratic kernel [95] for UCEA. We also give an FPT algorithm with a running time of $\mathcal{O}^*(1.47^k)$ for UCED, further enhancing the $\mathcal{O}^*(2^k)$ algorithm [106, 95]. We construct a subexponential algorithm for Uniform Cluster Edge Deletion on everywhere-α-dense graphs.

Maximum Minimal Degree-d-Modulator: The parameterized complexity of Degree-d-Modulator has been already studied [15, 38, 55, 97, 99] extensively. But there is no direct work on MaxMin Degree-d-Modulator, except for the case $d = 0$. As we have already noted that MaxMin Degree-0-Modulator is the same as MaxMin Vertex Cover. Fernau [57] presented fixed-parameter tractable (FPT) algorithms for the MaxMin Vertex Cover (MMVC), along with some results concerning its kernelization when parameterized by the solution size k. As noted in [57], the problem admits a kernel with at most k^2 vertices. Specifically, if there exists a vertex of degree at least k, then we can safely return a yes-instance. By using a similar technique, we show that MaxMin Degree-d-Modulator admits a vertex kernel of size $\mathcal{O}(dk^2 + d^2k)$.

It is worth noting that the MaxMin Vertex Cover (MMVC) problem is the dual of the well-known Minimum Independent Dominating Set. This correspondence arises from the observation that the complement of any minimal vertex cover forms an independent dominating set. We note that there exists related work [17, 111] on the dual of the MaxMin Degree-d-Modulator problem, although it has not been studied in a parameterized complexity setting.

MaxMin Vertex Cover can be solved in $\mathcal{O}^\star(c^k)$ time [114]. In route of the attempt to get a $\mathcal{O}^\star(c^k)$ for MaxMin Degree-d-Modulator, we prove that MaxMin Degree-d-Modulator is FPT parameterized by the treewidth of the graph. We prove that MaxMin Degree-d-Modulator can be solved in time $\mathcal{O}^\star(c^{\mathsf{tw}})$. For $d \in \{0, 1, 2\}$, it can be observed that given a degree-d-modulator of size at most k, one can construct a tree decomposition of

G with treewidth at most $k+2$. This implies that $\mathsf{tw}(G) = \mathcal{O}(k)$. Consequently, using the greedy algorithm and $\mathcal{O}^\star(c^{\mathsf{tw}})$ time algorithm for bounded treewidth, the MaxMin Degree-d-Modulator problem can be solved in $\mathcal{O}^\star(c^k)$ time for the special cases when $d \in \{0, 1, 2\}$.

Maximum Minimal Feedback Vertex Set: The MaxMin Feedback Vertex Set (MMFVS) was first demonstrated to be NP-complete for graphs with a maximum degree of 9 by Mishra and Sikdar [94]. This result was later improved by Dublois et al. [48], who showed NP-completeness for graphs with a maximum degree of 6. A polynomial-time approximation algorithm described in [48] implied the existence of a kernel of size $\mathcal{O}^\star(k^3)$. Further evidence suggesting that this kernel size may be optimal was provided later in [7]. Lampis et al. [87] showed that MMFVS can be solved in time $O(9.34^k)$. Additionally, it is worth noting that the problem is easily shown to be fixed-parameter tractable (FPT) when parameterized by treewidth, and even by clique-width, as the property of a set being a minimal feedback vertex set can be expressed in MSO_1 logic.

We consider the vertex cover as a parameter and show some results. First, we show that MMFVS can be solved in time $\mathcal{O}^\star\left(2^{\mathcal{O}(\mathsf{vc}(G)\log \mathsf{vc}(G))}\right)$. We complement this result by showing that MaxMin FVS parameterized by $\mathsf{vc}(G)$ does not admit a polynomial compression unless coNP $\subseteq$ NP/poly. Note that $\mathsf{vc}(G)$ can be much larger than $\mathsf{opt}(G)$, for example in a cycle. So $\mathcal{O}^\star(c^k)$ FPT algorithm for parameterized by solution size is not implied by our algorithm for the parameter vertex cover. Finally, we show that if we allow a "small loss" then we can improve upon our result for vertex cover. That is, we design an FPT-approximation algorithm for MMFVS parameterized by $\mathsf{vc}(G)$. Let $\epsilon > 0$ be a fixed constant. Then, there exists an algorithm for MMFVS, that runs in time $2^{\mathcal{O}\left(\frac{\mathsf{vc}(G)}{\epsilon}\right)} n^{\mathcal{O}(1)}$ and returns a minimal feedback vertex set of size at least $(1-\epsilon)\mathsf{opt}(G)$.

Lampis et al. [87] obtained an algorithm running in time $\mathsf{tw}^{\mathcal{O}(\mathsf{tw})} n^{\mathcal{O}(1)}$, which generalizes our $\mathsf{vc}^{\mathcal{O}(vc)} n^{\mathcal{O}(1)}$. They also showed that the existence of an algorithm running in time $\mathsf{vc}^{o(\mathsf{vc})} n^{\mathcal{O}(1)}$ is ruled out by the ETH (and hence also the existence of a $\mathsf{tw}^{o(\mathsf{tw})} n^{\mathcal{O}(1)}$ algorithm).

MaxMin Separation with Uncut Constraints: We observe that the MaxMin Multiway Cut-Uncut generalizes the MaxMin Multiway Cut, which in turn generalizes the MaxMin st-Cut. It is known that MaxMin st-Cut is NP-hard even when restricted to planar bipartite graphs [47]. Since both MaxMin Multiway Cut and MaxMin Multiway Cut-Uncut generalize this problem, it follows that they are also NP-hard on planar bipartite graphs. Similarly, when $r = 2$ and $\lambda(v) = \{1, 2\}$ for all $v \in V(G)$, the MaxMin

List Allocation coincides with the classical Maximum Cut, which was shown to be NP-hard even when restricted to graphs of maximum degree 3 by Yannakakis [113]. Hence, MaxMin List Allocation is also NP-hard. Furthermore, the MaxMin Subset FVS generalizes the MaxMin Feedback Vertex Set, which has been shown to be NP-hard on graphs of maximum degree 6 by Dublois et al. [48], who also present an approximation algorithm with ratio $n^{\frac{2}{3}}$ and proved that this is optimal unless P=NP. It follows that MaxMin Subset FVS is NP-hard as well.

We first remark that determining if there is any feasible solution for (MaxMin) Multiway Cut-Uncut is NP-hard, even when $|T_i| = 2$ for each $i \in \{1, \ldots, p\}$. Indeed, the feasibility question is equivalent to the Vertex Disjoint Paths. Thus, the NP-hardness of Vertex Disjoint Paths [83] imply that the feasibility version of (MaxMin) Multiway Cut-Uncut is NP-hard. Therefore, the MaxMin Multiway Cut-Uncut is para NP-hard when parameterized by solution size (as the instance with $k = 0$ is equivalent to the feasibility question), even when $|T_i| \leq 2$, for each $i \in \{1, \ldots, p\}$.

For parameterized tractability, the next interesting parameter is p (or even $k + p$) as at least the feasibility question, equivalently, Vertex Disjoint Paths is FPT parameterized by p [104, 105]. The feasibility version of the problem (that is the case when $k = 0$) turns out to be para-NP-hard even with the parameterization p, as already for $p = 2$, the feasible version of (MaxMin) Multiway Cut-Uncut is equivalent to the NP-hard 2-Disjoint Connected Subgraphs [110]. In the 2-Disjoint Connected Subgraphs, the input is an undirected graph G and two terminal sets $T_1, T_2 \subseteq V(G)$, $T_1 \cap T_2 = \emptyset$, and the question is to determine if the graph has two vertex-disjoint connected subgraphs, each containing T_1 and T_2 respectively. The 2-Disjoint Connected Subgraph is NP-hard even when one of the sets $|T_1| = 2$ (but $|T_2|$ is arbitrarily large). Thus, we conclude that MaxMin Multiway Cut-Uncut is para-NP-hard even with respect to the combined parameter $k + p$. This brings us to the main algorithmic part of our work. We show that MaxMin Multiway Cut-Uncut is FPT parameterized by $k + p$ when $\max_{i \in \{1,\ldots,p\}} |T_i| \leq 2$.

We establish the para-NP-hardness of MaxMin Multiway Cut-Uncut using the notion of feasible solutions of MaxMin Multiway Cut-Uncut. A similar technique can be used to derive hardness results for MaxMin Multiway Separator-Unseparator. We observe that $(G, (T_1, T_2))$ is a yes-instance of the 2-Induced Disjoint Paths if and only if there is a feasible solution to MaxMin Multiway Separator-Unseparator. It

is known that 2-Induced Disjoint Paths is NP-hard [16, 53]. Consequently, deciding if there is a feasible solution to MaxMin Multiway Separator-Unseparator is NP-hard even if $p = 2$, $k = 1$ and $|T_1| = |T_2| = 2$. Hence we conclude that MaxMin Multiway Separator-Unseparator is para-NP-hard parameterized by p and k combined even if $|T_1| = |T_2| = 2$.

We show that MaxMin Subset FVS is fixed-parameter tractable parameterized by k. We provide a parameterized reduction from MaxMin Multiway Separator to MaxMin Subset FVS, which shows that MaxMin Multiway Separator is fixed-parameter tractable parameterized by k. This generalizes the result provided by Gaikwad, Kumar, Maity, Saurabh & Sharma [66] who showed that the MaxMin st-Separator is FPT.

The List Allocation was introduced by Kim, Paul, Sau, and Thilikos [85], who showed that it is fixed-parameter tractable (FPT) when parameterized by $k = \sum_{1 \leq i < j \leq r} \alpha_{ij}$. Chandrasekaran and Wang [32] studied a relaxed variant of the problem, where the list assignment is uniform (i.e., $\lambda(v) = [r]$ for all vertices), and the objective is to bound the maximum number of edges crossing between any two parts by a global threshold α. They proved that this variant is W[1]-hard when parameterized by α, but also designed an FPT approximation algorithm for the problem. We show that MaxMin List Allocation is W[1]-hard parameterized by $\alpha_{\max} := \max_{1 \leq i < j \leq r} \alpha_{ij}$ by providing a parameterized reduction from the Biclique. We note that MaxMin List Allocation generalizes the Maximum Cut for the case $r = 2$. We know that Maximum Cut is NP-hard. Hence MaxMin List Allocation is para-NP-hard parameterized by r. Therefore, the interesting questions remain whether the problem is FPT when parameterized by $\alpha_{\max}$ only. We provide an FPT algorithm parametrized by $r + \alpha_{\max}$ by using color coding technique.

Vertex Splitting Problems: Planar Graph-Inclusive Vertex Splitting is known to be NP-complete [52]. Nöllenburg et al. [100] gave an FPT algorithm to solve Planar Graph-Inclusive Vertex Splitting. Baumann et al. [10] showed that $\mathcal{F}$-Inclusive Vertex Splitting is FPT parameterized by the number of splits when either restricted to input graphs of bounded treewidth and properties characterizable by monadic second-order graph logic, or alternatively, for minor-closed class $\mathcal{F}$. It is also known that Pathwidth-One Inclusive Vertex Splitting is FPT [10]. s-Club Cluster-Inclusive Vertex Splitting [4] is NP-complete and is shown to be FPT. Very recently, Abu-Khzam and Thoumi [5] showed that Claw-Free Exclusive Vertex Splitting is NP-complete in

general, while admitting a polynomial-time algorithm when the input graph has maximum degree four.

There are some interesting results for $\mathcal{F}$-Inclusive Vertex Splitting for some graph classes $\mathcal{F}$ in the thesis [58] of Alexander. $\mathcal{F}$-Inclusive Vertex Splitting is NP-complete [60, 58] for various hereditary graph classes, including, but not limited to, cluster graphs, bipartite graphs, perfect graphs, graphs free of finite sets of (induced) cycles, graphs free of a single biconnected forbidden (induced) subgraph, graphs free of a set of triconnected forbidden (induced) subgraphs with bounded diameter, and graphs free of a set of 4-connected forbidden (induced) subgraphs. Forest-, Split-, and Threshold- Inclusive Vertex Splitting can be solved in polynomial-time [58]. Cluster-Inclusive Vertex Splitting [59, 58] is NP-hard and admits a linear kernel.

We investigate the classical complexity of $\mathcal{F}$-Vertex Splitting for different $\mathcal{F}$. We observe that $\mathcal{F}$-Decompositon and $\mathcal{F}$-Vertex Splitting are closely related. By using this observation, we show that Cycle Graph-Exclusive Vertex Splitting is polynomial-time solvable. The complexity of Cycle Graph-Inclusive Vertex Splitting remains open. By giving a reduction from K_3-Decomposition to Uniform Cycle Graph-Inclusive Vertex Splitting, we prove that Uniform Cycle Graph-Inclusive Vertex Splitting and Uniform Cycle Graph-Exclusive Vertex Splitting are NP-complete. We prove that Star Forest-Inclusive Vertex Splitting and Star Forest-Exclusive Vertex Splitting are NP-complete on cubic graphs and even in planar graphs of degree at most 3.

Firbas and Sorge [58, 60] showed that Bipartite-Inclusive Vertex Splitting is NP-hard by giving a reduction from 2-Subdivided Cubic Vertex Cover. We prove that Bipartite-Inclusive Vertex Splitting and Bipartite Vertex Deletion are equivalent problems. This gives an alternate approach to prove NP-hardness of Bipartite-Inclusive Vertex Splitting. So, the existence of a kernel and FPT algorithm for Bipartite Vertex Deletion provides the existence of a kernel and FPT algorithm for Bipartite-Inclusive Vertex Splitting. We conclude the same for Bipartite-Exclusive Vertex Splitting as we show that Bipartite-Exclusive Vertex Splitting and Bipartite Vertex Deletion are equivalent problems.

Bicluster-Vertex Splitting has been stated as an open problem in [14]. We consider a slightly different problem, where the objective is to get a uniform bicluster graph instead of

a bicluster graph. In this thesis, we prove that Uniform Bicluster-Inclusive Vertex Splitting and Uniform Bicluster-Exclusive Vertex Splitting are NP-complete. We provide a *partial proof* of a conjecture proposed in Alexander's PhD thesis [58]. Establishing this conjecture could aid in proving the NP-completeness of the $\mathcal{F}$-Vertex Splitting problem for various classes $\mathcal{F}$. In the process of attempting to prove the conjecture, we established the NP-completeness of the Sigma Γ-Cover problem for a specific class, which may be of independent interest.

1.6 Overview of the Thesis

In Chapter 2, we establish the foundational terminology and notational conventions used throughout this thesis. We begin with a brief introduction to classical and parameterized complexity theory. In the context of parameterized complexity, we formally define parameterized problems, fixed-parameter tractable (FPT) algorithms, and kernelization. We also present several key techniques used to demonstrate that a problem belongs to the class FPT and introduce the W-hierarchy to illustrate intractability within this framework. A dedicated section focuses on structural graph parameters considered in this thesis. Subsequently, we provide a formal definition of Monadic Second-Order logic (MSO). Finally, we explore various variants of vertex splitting with an example.

In Chapter 3, we investigate the parameterized complexity of the Uniform Cluster Graph Modification problems. This chapter is based on the articles [64, 65]. We obtain kernels with $\mathcal{O}(k^3)$, $4k$, $4k$, $\mathcal{O}(k^2)$, $6k$, and $5k$ vertices for UCVD, UCEVS, UCIVS, UCEE, UCED, and UCEA, respectively. We design FPT algorithms with running times $\mathcal{O}(2^k)$ for UCVD and $\mathcal{O}(1.47^k)$ for UCED, and give a subexponential-time algorithm for UCED on everywhere-α-dense graphs. Parameterized by vertex cover, we present kernels of size $3\mathsf{vc}(G)$ and $2\mathsf{vc}(G)$ for UCVD and UCED, respectively. We also provide a kernel of size $3\mathsf{fvs}(G)+1$ for UCED when parameterized by feedback vertex set. Furthermore, we show that UCVD, UCEA, and UCEE are FPT when parameterized by vertex cover, and that UCEE admits an XP algorithm when parameterized by treewidth.

In Chapter 4, we study Degree d-Modulator. We show that Degree d-Modulator has a kernel with $\mathcal{O}(dk^2 + d^2k)$ vertices. We show that the Degree-d Modulator can be solved in $\mathcal{O}^\star(c^{\mathsf{tw}})$ time using a dynamic programming algorithm over a tree decomposition.

In Chapter 5, we focus on MaxMin Feedback Vertex Set parameterized by vertex cover number. This chapter is based on the article [67]. We show that MaxMin FVS can be solved in time $\mathcal{O}^\star(2^{\mathcal{O}(\mathsf{vc}(G)\log \mathsf{vc}(G))})$. We prove that MaxMin FVS parameterized by $\mathsf{vc}(G)$ does not admit a polynomial compression unless coNP $\subseteq$ NP/poly. Finally, we propose an FPT-approximation scheme parameterized by $\mathsf{vc}(G)$.

In Chapter 6, we investigate the parameterized complexity of three MaxMin separation problems: Maximum Minimal Multiway Cut Uncut (MaxMin Multiway Cut Uncut), Maximum Minimal List Allocation (MaxMin List Allocation) and Maximum Minimal Subset Feedback Vertex Set (MaxMin Subset FVS). We first give a technical overview of some of the proofs given in the chapter. We prove that MaxMin Multiway Cut-Uncut is para-NP-hard parameterized by solution size even when $|T_i| \leq 2$. Then we observe that MaxMin Multiway Cut-Uncut is para-NP-hard parameterized by p and k combined. In contrast, we prove that MaxMin Multiway Cut-Uncut is FPT parameterized by $p + k$ when $|T_i| \leq 2\ \forall i \in [p]$. We also observe that MaxMin Multiway Separator-Unseparator is para-NP-hard parameterized by p and k combined even if $|T_1| = |T_2| = 2$. Then we show that MaxMin Subset Feedback Vertex Set is FPT parameterized by solution size. The last section of the chapter is dedicated to MaxMin List Allocation. We first show that there exists a randomized algorithm that, given an instance of MaxMin List Allocation, in time $r^{\binom{r}{2}\alpha_{\max}} \cdot n^{\mathcal{O}(1)}$ either reports a failure or finds a r-colorful certificate of correctness of the given instance. Moreover, if the algorithm is given a yes instance, it returns a solution with constant probability. Finally, we prove that MaxMin List Allocation is W[1]-hard parameterized by $\alpha_{\max}$.

In Chapter 7, we analyze the classical complexity of $\mathcal{F}$-Vertex Splitting for various graph classes $\mathcal{F}$. This chapter is based on the article [68]. We show that Cycle Graph-Exclusive Vertex Splitting is solvable in polynomial time. In contrast, we prove that the inclusive and exclusive variants of $\mathcal{F}$-Vertex Splitting are NP-complete for the following classes: uniform cycle graphs, star forests, uniform biclusters, and bipartite graphs. Additionally, we establish the equivalence between Bipartite-Exclusive Vertex Splitting and Bipartite Vertex Deletion. Finally, we provide a partial proof of a conjecture from Alexander's PhD thesis [58].

In Chapter 8, we summarize our findings and present several open problems that emerge from our study.

Chapter 2

Preliminaries

This chapter introduces the notations and definitions used throughout the thesis. We begin by outlining the notations relevant to graph theory. Next, we present the key definitions and primary results from the field of classical and parameterized complexity theory. We then provide a brief overview of Monadic Second Order Logic on graphs, followed by a discussion of some of the structural parameters considered in this thesis. Finally, the last section of this chapter introduces the notion of vertex splitting.

2.1 Graph Theory

In this thesis, we focus on simple graphs with a finite number of vertices. For standard notations and definitions in graph theory, we refer to [112]. For an undirected graph G, $V(G)$ represents the set of vertices and $E(G)$ represents the set of edges of the graph G. Two vertices u and v in $V(G)$ are said to be adjacent if there exists an edge uv in $E(G)$. The neighborhood of a vertex v, denoted by $N_G(v)$, is the set of vertices adjacent to v, and its degree $d_G(v)$ is defined as $|N_G(v)|$. The subscript in the neighborhood and degree notation may be omitted when the graph being referenced is clear from the context.

Let $U \subseteq V$ be a subset of vertices of G and $F \subseteq \binom{V}{2}$ be a subset of pairs of vertices of G. The subgraph induced by $U \subseteq V$ is denoted by $G[U]$. We define $G - U = G[V \setminus U]$, $G - F = (V, E \setminus F)$, $G + F = (V, E \cup F)$ and $G \triangle F = (V, E \triangle F)$. Here, $E \triangle F$ is the

symmetric difference between E and F. If $U = \{u\}$ and $F = \{e\}$ then we simply write $G - u$, $G - e$ and $G + e$ for $G - U$, $G - F$ and $G + F$, respectively. For a set of edges F, the set $V(F)$ refers to the collection of all endpoints of the edges in F. For a fixed graph H, $\text{Free}_{\simeq}(H) := \{G \mid \text{no induced subgraph of } G \text{ is isomorphic to } H\}$.

A graph G is *d-regular* if every vertex has degree d. The graph on n vertices in which every vertex is adjacent to every other vertex is called the *complete graph* and denoted by K_n. Note that K_n is $(n-1)$-regular. A graph G is called *bipartite* if its vertex set can be partitioned into two subsets V_1 and V_2, such that every edge connects a vertex from V_1 to a vertex from V_2. If each vertex in V_1 is adjacent to every vertex in V_2, the graph is referred to as complete bipartite and is denoted by $K_{m,n}$, where $m = |V_1|$ and $n = |V_2|$.

A *path* $P = (v_1, v_2, \ldots, v_\ell)$ is a sequence of distinct vertices where every consecutive pair of vertices is adjacent. A path with ℓ vertices is denoted by P_ℓ. A *cycle* is a sequence $(v_1, v_2, \ldots, v_\ell, v_1)$ of vertices such that $(v_1, v_2, \ldots, v_\ell)$ is a path and $v_\ell v_1$ is an edge. A cycle with ℓ vertices is denoted by C_ℓ. A graph is said to be *connected* if there exists a path between every pair of vertices. If a graph is not connected, it is referred to as *disconnected*. The maximal connected subgraphs of a graph are known as *components*. A *tree* is a connected graph with no cycles. A *rooted tree* is a tree where one vertex is specifically designated as the root. The *height of a rooted tree* is the length of the longest path from the root to any vertex. A vertex of degree one is called a *pendant vertex*. A pendant vertex in a tree is called a *leaf*. A *star graph* is a complete bipartite graph $K_{1,n}$, where one part has a single vertex (the central vertex) and the other part contains n vertices (the leaves). A *forest* is a graph containing no cycles.

A *clique* is a subset of vertices in an undirected graph where every pair of distinct vertices is adjacent. In contrast, an *independent set* is a subset of vertices in which no two vertices are adjacent to each other. A *cluster graph* is a graph where every component is a clique. Observe that a graph is a cluster graph if and only if it does not have an induced P_3, that is, an induced path on three vertices.

A graph on n vertices is *α-dense* if it has $\alpha n^2/2$ edges. It is *everywhere-α-dense* if the minimum degree is αn. We abbreviate $\Omega(1)$-dense as *dense* and everywhere-$\Omega(1)$-dense as *everywhere-dense*.

For two positive integers q and k, a graph G is said to be *(q,k)-unbreakable* if *no* vertex

set of size at most k can separate two vertex subsets of size at least q each. In other words, for every separation of G of order at most k, at least one side of the separation has fewer than q vertices outside the separator. We say that a graph is *unbreakable* if it is (q, k)-unbreakable for some values of q and k.

A pair (X, Y), where $X \cup Y = V(G)$, is called a *separation* of G if there are no edges between $X \setminus Y$ and $Y \setminus X$, i.e., $E(X \setminus Y, Y \setminus X) = \emptyset$. The *order* of the separation (X, Y) is defined as $|X \cap Y|$.

We note that if there exists a separation (X, Y) of order at most k such that $|X \setminus Y| \geq q$ and $|Y \setminus X| \geq q$, then G is (q, k)-*breakable*, and the separation (X, Y) is called a *witnessing separation* for the (q, k)-breakability of G. Otherwise, G is (q, k)-*unbreakable*.

The *adjacency matrix* A_G of G, with respect to a listing of the vertices, is the $n \times n$ zero-one matrix with 1 as its (i, j) entry when v_i and v_j are adjacent, and 0 otherwise. Adjacency matrices of undirected graphs are always symmetric. It means the adjacency matrix of a graph with n vertices has n number of real eigenvalues. It is known [43, 70] that the adjacency matrix A_G has at most two distinct eigenvalues if and only if G is a disjoint union of equal-sized cliques.

2.2 Classical and Parameterized Complexity Theory

We assume a foundational understanding of computational complexity theory, including familiarity with the complexity classes P and NP, polynomial-time many-one reductions, and the concepts of problem hardness and completeness for a given class. We will address some of the mentioned topics very briefly. For a thorough introduction to classical complexity theory, refer to [36].

Problems in the set P are those problems that have polynomial time algorithms, and are computationally the easiest problems to solve. The NP-complete problems are the "polynomially hardest" problems in the class NP, and are all *polynomially intractable*, making them a special focus in the theory of classical complexity. Let us understand the concept of *hardness* between problems. Consider two decision problems, A and B, and an algorithm $\mathcal{A}$ that can convert an instance of A into an equivalent instance of B in polynomial time. If B

can be solved in polynomial time, then it follows that A can also be solved in polynomial time: we first use algorithm $\mathcal{A}$ to transform the instance of A into an equivalent instance of B, and then we solve B, with both steps taking polynomial time. However, the existence of a polynomial-time algorithm for A does not necessarily imply the existence of a polynomial-time algorithm for B. Thus, we say B is *polynomially harder* than A, denoted as $A \leq_P B$. The algorithm $\mathcal{A}$ is referred to as a *polynomial-time reduction* from A to B. We consider the problems A and B to be *polynomially equivalent* if both $A \leq_P B$ and $B \leq_P A$ hold true, which is represented as $A =_P B$.

The time required by an algorithm is a key factor in determining its efficiency. For many problems, there are algorithms with a time complexity that is polynomial in the input size. Many real-world problems are NP-Hard, which means they are very difficult to solve efficiently, and we do not expect to find fast algorithms to solve them for large inputs. Parameterized complexity provides a special approach to deal with these challenging problems. The area of parameterized complexity theory was first systematically explored by Downey and Fellows [46] in 1999. Parameterized Complexity theory provides a more detailed view of the computational complexity of problems. Central notions in this paradigm are *fixed-parameter tractability* and *kernelization*. For a comprehensive introduction to parameterized complexity theory, we refer to [40].

A *parameterized problem* is a classical problem with an additional integer associated with each instance of the problem. Formally, we define it as follows.

Definition 10. *A parameterized problem is a language $\Pi \subseteq \Sigma^* \times \mathbb{N}$, where Σ^* is a fixed, finite alphabet. For an instance $(I, k) \in \Sigma^* \times \mathbb{N}$, the integer k is called the parameter.*

The parameter is expected to be smaller than the entire input. A common parameter is a bound on the size of an optimal solution to the problem instance.

Definition 11. *A parameterized problem $\Pi \subseteq \Sigma^* \times \mathbb{N}$ is called fixed-parameter tractable if there exists an algorithm $\mathcal{A}$ (called a fixed parameter tractable algorithm), a computable function $f : \mathbb{N} \to \mathbb{N}$, and a constant c such that, given $(I, k) \in \Sigma^* \times \mathbb{N}$, the algorithm $\mathcal{A}$ correctly decides whether $(I, k) \in \Pi$ in time bounded by $f(k) \cdot (|I| + k)^c$.*

The complexity class containing all fixed-parameter tractable problems is called FPT.

Kernelization is a method used to simplify problems by reducing their size while still keeping the main characteristics intact. It is also used to show that a parameterized problem is in FPT.

Definition 12. *A kernelization algorithm, or simply a kernel, of a parameterized language* $\Pi \subseteq \Sigma^* \times \mathbb{N}$ *is an algorithm that takes as input an instance* $(I, k) \in \Sigma^* \times \mathbb{N}$*, and in time polynomial in* $|I| + k$*, returns another instance* (I', k') *such that:*

- $|I'| + k' \leq g(k)$ *for some computable function* $g(\cdot)$*, and*
- $(I, k) \in \Pi$ *if and only if* $(I', k') \in \Pi$*.*

Two instances (I, k) and (I', k') are said to be equivalent instances of Π when $(I, k) \in \Pi$ if and only if $(I', k') \in \Pi$. Depending on whether the function $g(\cdot)$ is linear, polynomial, or exponential, the problem is said to have a linear, polynomial, or exponential kernel, respectively.

If we have a kernelization algorithm for a problem for which there is some algorithm (with any running time) to decide whether (I, k) is a yes-instance, then clearly the problem is FPT, as the size of the reduced instance I' is simply a function of k (and independent of the input size). Interestingly, the converse is also true. So, we have the following theorem.

Theorem 1. *[40] A parameterized problem admits a kernelization algorithm if and only if it is fixed-parameter tractable.*

Any FPT problem admits an exponential kernel [40]. So a natural question for any FPT problem is whether it admits a polynomial kernel. However, not every problem that is FPT with respect to a specified parameter necessarily has a polynomial kernel.

We discuss the concept of polynomial compression, which serves as a generalization of the notion of kernels in parameterized complexity. Unlike kernels, which specifically aim to reduce the instance into the same language, polynomial compression allows reduction into a potentially different target language.

Definition 13. *A* polynomial compression *of a parameterized language* $Q \subseteq \Sigma^* \times \mathbb{N}$ *into a language* $R \subseteq \Sigma^*$ *is an algorithm that takes as input an instance* $(x, k) \in \Sigma^* \times \mathbb{N}$*, works in time polynomial in* $|x| + k$*, and returns a string* y *such that:*

- $|y| \leq p(k)$ *for some polynomial* $p(\cdot)$*, and*
- $y \in R$ *if and only if* $(x, k) \in Q$*.*

Note that a polynomial kernel is also a polynomial compression by treating the output kernel as an instance of the unparameterized version of Q*.*

Definition 14. *Let* $\Pi_1, \Pi_2 \subseteq \Sigma^* \times \mathbb{N}$ *be two parameterized problems. An algorithm* $\mathcal{A}$ *is called a* polynomial parameter transformation *from* Π_1 *to* Π_2 *if, given an instance* (I_1, k_1) *of* Π_1*,* $\mathcal{A}$ *works in polynomial time and outputs an instance* (I_2, k_2) *of* Π_2 *such that:*

- $|k_2| \leq p(k_1)$ *for some polynomial* $p(\cdot)$*, and*
- $(I_1, k_1) \in \Pi_1$ *if and only if* $(I_2, k_2) \in \Pi_2$*.*

Let Π_1 *be a problem known to not admit a polynomial compression. Then polynomial parameter transformation would help to transfer this hardness to* Π_2*.*

Theorem 2. *[40] Let* Π_1 *and* Π_2 *be two parameterized problems in* $\Sigma^* \times \mathbb{N}$*, and suppose there exists a polynomial parameter transformation from* Π_1 *to* Π_2*. If* Π_1 *is known to not admit a polynomial compression, then* Π_2 *does not admit a polynomial compression.*

Theorem 1 demonstrates that a kernelization algorithm is a powerful technique for designing a fixed-parameter tractable (FPT) algorithm. In the following, we will explore the concept of *parameterized reduction*, which is another important method used to establish that a problem is in FPT.

Definition 15. *Let* $\Pi_1, \Pi_2 \subseteq \Sigma^* \times \mathbb{N}$ *be two parameterized problems. A parameterized reduction from problem* Π_1 *to* Π_2 *is an algorithm that, given an instance* (x, k) *of* Π_1*, outputs an instance* (x', k') *of* Π_2 *such that*

- (x, k) *is a yes instance of* Π_1 *if and only if* (x', k') *is a yes instance of* Π_2*,*
- $k' \leq g(k)$ *for some computable function* g*, and*
- *the running time is* $f(k) \cdot |x|^{\mathcal{O}(1)}$ *for some computable function* f*.*

In classical complexity theory, we have seen the notion of polynomial-time reduction. We have seen that if there is a polynomial-time reduction from problem A to a problem B then if problem B can be solved in polynomial time then problem A also admits a polynomial time algorithm. Analogously, in parameterized complexity, the idea of parameterized reduction serves a similar purpose. A parameterized reduction ensures that the parameter in the transformed instance remains bounded by a function of the original parameter. This preservation is crucial, as it guarantees that the computational hardness associated with the parameter is maintained. Parameterized reductions thus provide a rigorous framework for analyzing and comparing the complexity of problems within the parameterized complexity theory. The following theorem is a straightforward consequence of the definition of parameterized reduction

Theorem 3. *[40] If there is a parameterized reduction from problem* Π_1 *to* Π_2 *and* Π_2 *is FPT, then* Π_1 *is FPT as well.*

Another widely used technique to demonstrate that a parameterized problem is fixed-parameter tractable (FPT) is the *branching technique*, also known as *bounded search tree*. Branching algorithms systematically explore a search tree, where each node represents a partial solution to the problem instance. At each branching point, the algorithm makes decisions based on the current state and the values of additional parameters, generating subproblems that correspond to alternative choices. This recursive process continues until a complete solution is found or the entire search space has been traversed. Despite their simplicity, branching algorithms often exhibit exponential worst-case time complexity due to the combinatorial explosion of choices at each decision point. The efficiency of such algorithms is heavily influenced by both the *branching factor* (i.e., the number of subproblems generated at each node) and the *depth* of the search tree.

We have presented the most fundamental algorithmic techniques for showing the fixed-parameter tractability of parameterized problems. Now our focus shifts to the complementary perspective, examining techniques for providing evidence that certain parameterized problems are unlikely to admit fixed-parameter tractable algorithms. We have already noted that if there is a parameterized reduction from problem Π_1 to Π_2 and the problem Π_2 is FPT, then problem Π_1 is also FPT. The existence of a parameterized reduction from problem Π_1 to problem Π_2 tells that problem Π_1 is at least as hard as problem Π_2. We note that if we assume that the problem Π_1 is not FPT then the problem Π_2 is not FPT as well.

Assuming, as a working hypothesis, that the CLIQUE is not fixed-parameter tractable, a parameterized reduction from CLIQUE to another problem provides evidence that the latter problem is also unlikely to be fixed-parameter tractable. There are different *levels* in the parameterized complexity of hard problems. We will not go into the details of *levels* as in this thesis, we are mainly interested in knowing whether a given problem admits an FPT algorithm or not. We refer to [40] for more details on the W-hierarchy. CLIQUE, which is not tractable by a fixed parameter, is equivalent to FPT $\neq$ W [1].

Typically, for a parameterized problem with parameter k, k reflects some structural property of the instance. In the next section, we will discuss some of the structural parameters that we consider in this thesis.

2.3 Structural Graph Parameters

In this section, we recall the definitions of the structural graph parameters discussed in this thesis.

Definition 16. ***(Vertex Cover):*** *A set $S \subseteq V(G)$ is a vertex cover of a graph $G = (V, E)$ if each edge in E has at least one endpoint in S. The size of the smallest vertex cover of G is called the vertex cover number of G, denoted by $\mathsf{vc}(G)$.*

Figure 2.1 shows an example of a graph G with $\mathsf{vc}(G) = 2$.

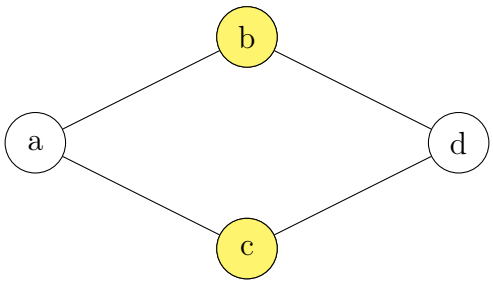


Figure 2.1: A simple graph G in which $\{b, c\}$ forms a minimum vertex cover. Every edge is incident to at least one of b or c, hence $\mathsf{vc}(G) = 2$.

Definition 17. ***(Feedback Vertex Set):*** *A feedback vertex set of a graph G is a set of vertices whose removal leaves G acyclic (without cycles). The minimum size of a feedback vertex set in G is called the feedback vertex set number of G, denoted by $\mathsf{fvs}(G)$.*

Figure 2.2 shows an example of a graph G with $\mathsf{fvs}(G)$=2.

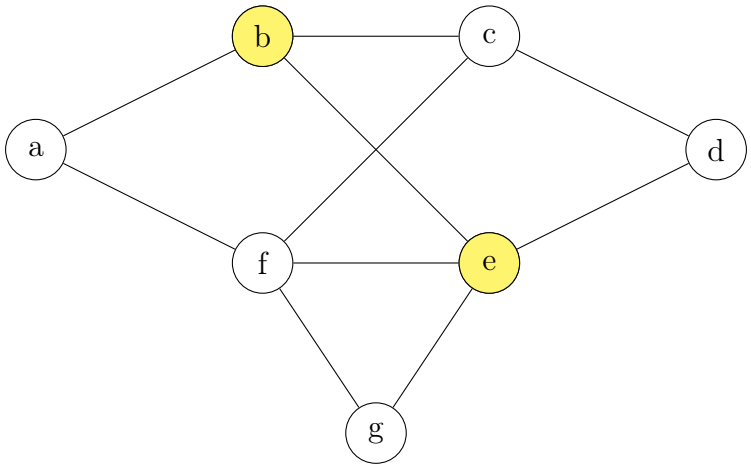


Figure 2.2: A graph G containing multiple cycles. Removing vertices b and e eliminates all cycles, leaving a tree. The set $\{b, e\}$ forms a minimum feedback vertex set, hence] $\mathsf{fvs}(G) = 2$.

The concept of tree decomposition was introduced by Robertson and Seymour [103]. Treewidth is a measure of how "tree-like" the graph is.

Definition 18. *A tree decomposition of a graph G is a pair $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$, where T is a tree whose every node t is assigned a vertex subset $X_t \subseteq V(G)$, called a bag, such that the following three conditions hold:*

(T1) $\bigcup_{t \in V(T)} X_t = V(G)$. *In other words, every vertex of G is in at least one bag.*

(T2) For every $uv \in E(G)$, there exists a node t of T such that the bag X_t contains both u and v.

(T3) For every $u \in V(G)$, the set $T_u = \{t \in V(T) : u \in X_t\}$, that is, the set of nodes whose corresponding bags contain u, induces a connected subtree of T.

The width of tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ equals $\max_{t \in V(T)} |X_t| - 1$. Figure 2.3 shows an example of a tree decomposition of width two of C_5.

To distinguish between the vertices of the decomposition tree T and the vertices of the graph G, we will refer to the vertices of T as *nodes*. Note that we can always assume that no two consecutive bags of a tree decomposition are equal since removing one of such bags does not violate any property of a tree decomposition.

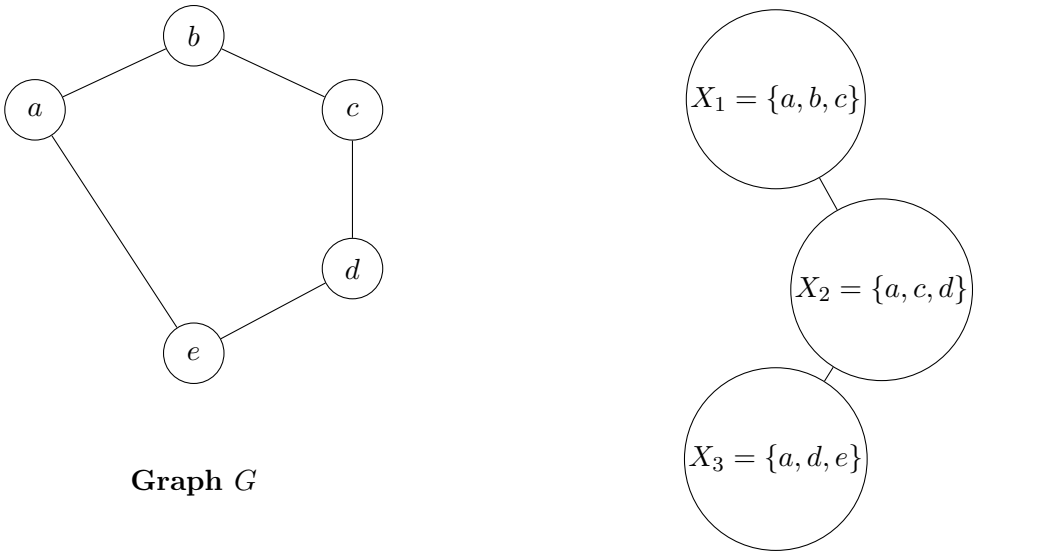


Figure 2.3: A tree decomposition of G satisfies the following conditions: (T1) every vertex of G appears in at least one bag, (T2) every edge of G is contained in some bag (for example, (c, d) lies in X_2, (d, e) lies in X_3), (T3) for each vertex, the set of bags containing it (for example, the bags containing d are X_2, X_3) forms a connected subtree.

Definition 19. ***(Treewidth):*** *The treewidth of a graph G, denoted by $\mathsf{tw}(\mathsf{G})$, is the minimum possible width of a tree decomposition of G.*

For a tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$, we distinguish one vertex r of T which will be the root of T. This introduces natural parent-child and ancestor-descendant relations in the tree T. We now introduce a "canonical" form of a tree decomposition, which will be helpful for the dynamic programming algorithms discussed in later chapters.

Definition 20. *A rooted tree decomposition $(T, \{X_t\}_{t \in V(T)})$ is called nice if the following conditions are satisfied:*

- *$X_r = \emptyset$ and $X_\ell = \emptyset$ for every leaf ℓ of T. In other words, all the leaves as well as the root contain empty bags.*
- *Every non-leaf node of T is of one of the following three types:*
  - ***Introduce node:*** *A node t with exactly one child t_0 such that $X_t = X_{t_0} \cup \{v\}$ for some vertex $v \notin X_{t_0}$. We say that v is introduced at t.*

- ***Forget node:*** *A node t with exactly one child t_0 such that $X_t = X_{t_0} \setminus \{w\}$ for some vertex $w \in X_{t_0}$. We say that w is forgotten at t.*
- ***Join node:*** *A node t with two children t_1 and t_2 such that $X_t = X_{t_1} = X_{t_2}$.*

It is known [40] that if a graph G admits a tree decomposition of width at most tw, then it also admits a nice tree decomposition of width at most tw, that has at most $\mathcal{O}(n \cdot \mathsf{tw})$ nodes

2.4 Monadic Second Order Logic (MSO)

Monadic Second Order Logic (MSO) is a logical language used to express properties of graphs and their components, such as vertices, edges, and subsets thereof. A formula φ in MSO is essentially a string composed of specific logical symbols and syntactic constructs defined by the formal grammar of the language. The syntax of *Monadic Second-Order Logic* (MSO) for graphs includes the standard logical connectives $\vee$ (*disjunction*), $\wedge$ (*conjunction*), $\neg$ (*negation*), $\implies$ (*implication*), $\iff$ (*logical equivalence*); variables representing vertices, edges, sets of vertices, and sets of edges; the quantifiers $\forall$ (*universal quantifier*) and $\exists$ (*existential quantifier*); and the following five types of atomic predicates:

1. $u \in U$, where u is a vertex variable and U is a vertex set variable;
2. $e \in E$, where e is an edge variable and E is an edge set variable;
3. $\mathbf{inc}(e, u)$, indicating that the edge e is incident to the vertex u;
4. $\mathbf{adj}(u, v)$, indicating that the vertices u and v are adjacent;
5. Equality between variables of the same type (i.e., vertices, edges, vertex sets, or edge sets).

Counting Monadic Second-Order Logic (CMSO) extends MSO by incorporating atomic formulas that express modular cardinality constraints on sets. Specifically, CMSO includes atomic formulas of the form:

$$\mathbf{card}_{q,r}(S),$$

which evaluates to true if and only if $|S| \equiv q \pmod{r}$, where S is a set variable (either of vertices or edges), and q, r are integers such that $0 \leq q < r$ and $r \geq 2$.

A detailed study of Counting Monadic Second-Order (CMSO) logic on graphs can be found in [40]. We now present a fundamental result from [91] that elegantly bridges the notions of fixed-parameter tractability (FPT), (q, k)-unbreakability, and Counting Monadic Second-Order logic (CMSO).

Theorem 4 ([91]). *Let ψ be a CMSO formula. For all $c \in \mathbb{N}$, there exists $s \in \mathbb{N}$, such that if there exists an algorithm that solves CMSO[ψ] on (s, c)-unbreakable graphs in time $\mathcal{O}(n^d)$ for some $d > 4$, then there exists an algorithm that solves CMSO[ψ] on general graphs in time $\mathcal{O}(n^d)$.*

2.5 Vertex Splitting

A *vertex split* is a graph modification where a vertex v is replaced by two new vertices, v_1 and v_2. Each edge that was originally incident to v is then reassigned to v_1 or v_2.

Abu-Khzam et al. [3] proposed two different vertex splitting operations:

- *Inclusive Vertex Splitting:* $N(v_1) \cup N(v_2) = N(v)$.
- *Exclusive Vertex Splitting:* $N(v_1) \cup N(v_2) = N(v)$ and $N(v_1) \cap N(v_2) = \emptyset$.

The key difference between these two modifications lies in their treatment of vertex neighborhoods. In Inclusive Vertex Splitting, v_1 and v_2 are allowed to share neighborhoods, while in Exclusive Vertex Splitting, v_1 and v_2 must have disjoint neighborhoods.

A significant limitation of both existing variants of vertex splitting is that they require $N(v_1) \cup N(v_2) = N(v)$. Bentert et al. [14] introduced a variant of vertex splitting where we do not have the restriction $N(v_1) \cup N(v_2) = N(v)$:

- *Permissible Vertex Splitting:* $N(v) \subseteq N(v_1) \cup N(v_2)$ and $N'(v) \subseteq N'(v_1) \cup N'(v_2)$.

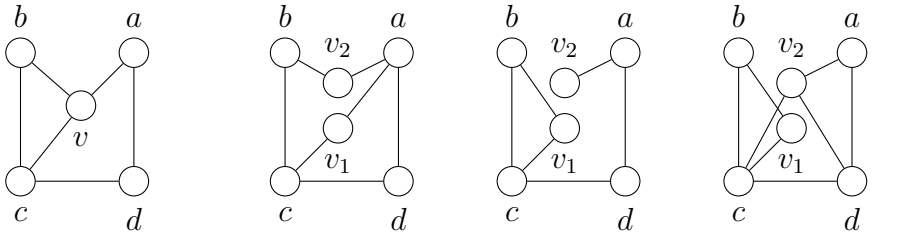


Figure 2.4: Variants of Vertex Splitting

Figure 2.4 shows an example with different vertex splitting operations.

If v is split into v_1 and v_2, then v_1 and v_2 are called the *descendants* of v. Conversely, v is called the *ancestor* of v_1 and v_2.

Every vertex split increases the number of vertices in a graph by exactly one. Typically, we are interested in carrying out more than one vertex split. To facilitate this, we introduce the concept of a splitting sequence.

Definition 21. *A splitting sequence of k splits is a sequence of graphs $G_0, G_1, \ldots, G_k$, such that G_{i+1} is obtainable from G_i via a vertex split for $i \in \{0, \ldots, k-1\}$.*

The notion of *descendant vertices* (resp. *ancestor vertices*) is extended in a transitive and reflexive way to splitting sequences.

Chapter 3

Uniform Cluster Graph Modifications

In clustering, many real-world challenges involve grouping objects or data points into well-defined clusters. By representing these challenges as graphs, where nodes represent objects and edges represent relationships, we can utilize graph modification techniques to enforce structural properties, such as equal-sized clusters or clusters containing highly similar elements. The ability to edit graphs—by deleting or adding edges, or by removing vertices—facilitates more accurate or computationally feasible clustering, particularly when the initial data does not naturally conform to ideal clusters.

In this thesis, we study a relatively new variant of clustering problem. We focus on computational problems where the goal is to modify a given graph through vertex deletions, vertex splittings, edge deletions, edge additions, or edge edits to transform it into a collection of equal-sized cliques, which we refer to as *Uniform Cluster graphs*. Specifically, we investigate the Uniform Cluster Vertex Deletion (UCVD) problem, where the task is to delete at most k vertices to achieve this transformation. Similarly, we consider the Uniform Cluster Vertex Splitting (UCVS) problem, where the task is to decide whether there is a sequence of at most k vertex splits to get a uniform cluster graph. Additionally, we consider the edge-based variants: Uniform Cluster Edge Deletion (UCED), Uniform Cluster Edge Addition (UCEA), and Uniform Cluster Edge Editing (UCEE), where the objective is to apply a bounded number of these operations to yield a graph that becomes a Uniform Cluster graph. This chapter is based on the articles [64, 65]. We formally define the problems studied in this chapter below.

Uniform Cluster Vertex Deletion (UCVD)
Input: An undirected graph $G = (V, E)$ and a positive integer k.
Question: Is there a subset $S \subseteq V$ with $|S| \leq k$ such that $G - S$ is a collection of equal-sized cliques?

Uniform Cluster Inclusive Vertex Splitting (UCIVS)
Input: An undirected graph $G = (V, E)$ and a positive integer k.
Question: Is there a sequence of at most k inclusive vertex splits that transforms G into a collection of equal-sized cliques?

Uniform Cluster Exclusive Vertex Splitting (UCEVS)
Input: An undirected graph $G = (V, E)$ and a positive integer k.
Question: Is there a sequence of at most k exclusive vertex splits that transforms G into a collection of equal-sized cliques?

Uniform Cluster Edge Editing (UCEE)
Input: An undirected graph $G = (V, E)$ and a positive integer k.
Question: Is there a subset $F \subseteq \binom{V(G)}{2}$ with $|F| \leq k$ such that $G \triangle F = (V, E \Delta F)$ is a collection of equal-sized cliques?

Uniform Cluster Edge Deletion (UCED)
Input: An undirected graph $G = (V, E)$ and a positive integer k.
Question: Is there a subset $F \subseteq E$ with $|F| \leq k$ such that $G - F = (V, E \setminus F)$ is a collection of equal-sized cliques?

Uniform Cluster Edge Addition (UCEA)
Input: An undirected graph $G = (V, E)$ and a positive integer k.
Question: Is there a subset $F \subseteq \binom{V(G)}{2}$ with $|F| \leq k$ such that $G + F = (V, E \cup F)$ is a collection of equal-sized cliques?

3.1 Kernelization and FPT algorithm for UCVD parameterized by solution size

In this section, we provide kernelization and fixed-parameter tractable (FPT) algorithms for the Uniform Cluster Vertex Deletion (UCVD).

Theorem 5. *The* UCVD *parameterized by solution size admits a kernel of size* $\mathcal{O}(k^3)$.

Proof. A cluster graph is a graph in which every connected component is a clique. Note that a graph is a cluster graph if and only if it does not contain an induced P_3. The first step of the kernelization process is to compute a maximal set $\mathcal{P}_3$ of vertex-disjoint induced P_3s in G. This can be accomplished using the greedy algorithm shown below:

Algorithm 1

Require: A graph $G = (V, E)$.
Ensure: A maximal set $\mathcal{P}_3$ of vertex-disjoint induced P_3 in G.
1: $\mathcal{P}_3 = \emptyset$
2: **while** G has an induced P_3 **do**
3: Identify an induced $P_3 = (u, v, w)$ in G
4: $\mathcal{P}_3 = \mathcal{P}_3 \cup \{(u, v, w)\}$
5: $G = G - \{u, v, w\}$
6: **end while**
7: return $\mathcal{P}_3$

At the end, if $|\mathcal{P}_3| > k$, we have a no-instance. So we assume that $|\mathcal{P}_3| \leq k$, and let S be the vertices of $\mathcal{P}_3$. We have $|S| \leq 3k$. Let us denote the set of cliques of $G - S$ by $\mathcal{C}$.

We begin with the following reduction rule, which provides two sufficient conditions for a vertex $s \in S$ to be included in a solution.

Reduction Rule UCVD 1. *Let $s \in S$ be a vertex such that one of the following conditions hold:*

- *s has neighbors in at least $k + 2$ distinct cliques in $\mathcal{C}$, or*
- *s has at least $k + 1$ neighbors in a clique $C \in \mathcal{C}$ and more than k neighbors in cliques other than C.*

Then, delete s from G and decrement the parameter k by 1. The new instance becomes $(G - s, k - 1)$.

Lemma 6. *Reduction Rule UCVD 1 is safe.*

Proof. Let X be a uniform cluster vertex deletion set of G of size at most k. For the sake of contradiction, assume that $s \notin X$. In all cases outlined by Reduction Rule 1, the graph $G - X$ will contain a P_3 containing s, violating the structure of a (uniform) cluster graph. This contradicts the assumption that X is a valid uniform cluster vertex deletion set. Hence, s must belong to any solution for (G, k), and it is safe to delete s and decrement k by 1. □

Next, we partition the set of cliques in $\mathcal{C}$ based on whether they have neighbours in S or not. We define:

$$\mathcal{C}_0 = \{C \in \mathcal{C} : \text{no vertex in } C \text{ has a neighbour in } S\}$$

and

$$\mathcal{C}_1 = \{C \in \mathcal{C} : \text{some vertex in } C \text{ has a neighbour in } S\}.$$

An r-clique is a clique of size r. Now, we have the following simple rule.

Reduction Rule UCVD 2. *If for some $r \in \{1, 2, \ldots, n\}$ there are more than $k+1$ r-cliques in $\mathcal{C}_0$, then remove all but $k + 1$ r-cliques from G.*

Lemma 7. *Reduction Rule UCVD 2 is safe.*

Proof. Suppose (G, k) is a yes-instance and X is a uniform cluster vertex deletion set of G of size at most k. If for some $r \in \{1, 2, \ldots, n\}$ there are more than $k + 1$ r-cliques in $\mathcal{C}_0$, then we claim that r is the size of all cliques in $G - X$. For the sake of contradiction, let us assume that $r' < r$ is the size of all cliques in $G - X$. In that case, we need to delete at least one vertex from each r-clique. This means we would have to add at least $k + 1$ vertices in X, which contradicts the assumption that X is of size at most k. Thus, we conclude that r is the size of all cliques in $G - X$. Since the solution size is at most k, this allows us to remove all but $k + 1$ r-cliques from G. □

Reduction Rule UCVD 3. *If $\mathcal{C}_0$ contains cliques of more than $k + 1$ distinct sizes then return a no instance.*

This is correct because by deleting at most k vertices we can alter the sizes of at most k cliques. Therefore, in this case, there can not exist a uniform cluster vertex deletion of size at most k.

Lemma 8. *After applying Reduction Rules 1, 2 and 3, the set $\mathcal{C} = \mathcal{C}_0 \cup \mathcal{C}_1$ contains at most $\mathcal{O}(k^2)$ cliques.*

Proof. Due to Reduction Rule 2 and 3, we know that $|\mathcal{C}_0| \leq (k+1)(k+1)$. After the exhaustive application of Reduction Rule 1, each vertex of S can have neighbours in at most $k+1$ cliques of $\mathcal{C}_1$. Since $|S| \leq 3k$ and each vertex of S can have neighbours in at most $k+1$ cliques of $\mathcal{C}_1$, there are at most $3k(k+1)$ cliques in $\mathcal{C}_1$. Therefore, we have $|\mathcal{C}_1| \leq 3k(k+1)$. Combining the two, we get

$$|\mathcal{C}| = |\mathcal{C}_0| + |\mathcal{C}_1| \leq (k+1)(k+1) + 3k(k+1) = 4k^2 + 5k + 1.$$

□

Now, we consider two cases based on the maximum size of a clique in $\mathcal{C}$. Let $\omega(\mathcal{C}) = \max_{C \in \mathcal{C}}\{|C|\}$.

Case 1: $\omega(\mathcal{C}) < 8k$

In this case, we obtain a kernel of size:

$$|S| + \sum_{C \in \mathcal{C}} |C| < 3k + 8k(4k^2 + 5k + 1) = \mathcal{O}(k^3).$$

Case 2: $\omega(\mathcal{C}) \geq 8k$

Let C_0 be a clique in $\mathcal{C}$ such that $\omega(\mathcal{C}) = |C_0|$. Then C_0 becomes a clique of size at least $|C_0| - k \geq 7k$ in $G - X$, where X is a solution of size at most k. In this case, the resulting graph $G - X$ must be a collection of r-cliques, where $r \geq |C_0| - k \geq 7k$.

Reduction Rule UCVD 4. *If there exist C_1 and $C_2 \in \mathcal{C}$ such that $|C_1| - |C_2| > 4k$, then delete all the vertices in C_2 and decrement the parameter k by $|C_2|$. The new instance is $(G - C_2, k - |C_2|)$.*

Lemma 9. *Reduction Rule UCVD 4 is safe,*

Proof. For the sake of contradiction, let us assume that $v \in C_2$ does not belong to some solution X of size at most k. First, since C_2 is a clique, v has at most $|C_2| - 1$ neighbours in C_2 in the resulting graph. Second, v can have at most $3k$ neighbours in S. Thus, v can be part of a clique of size at most $(|C_2| - 1) + 3k + 1 = |C_2| + 3k$.

On the other hand, there always exist a vertex $u \in C_1$ such that u is not contained in X. Clearly, u will have at least $|C_1| - k - 1$ neighbours in $G - X$. Therefore, u must be contained in a clique of size at least $|C_1| - k > |C_2| + 4k - k = |C_2| + 3k$. Therefore, u and v can never be contained in equal-sized cliques. This contradicts the fact that $G - X$ is a collection of equal sized-cliques. Therefore, we must include all vertices of C_2 in every uniform cluster vertex deletion set X of G of size at most k. □

Observation 3.1.1. *After applying Reduction Rule UCVD 4, we get that* $\min_{C \in \mathcal{C}} |C| \geq 4k$.

This is true because we have assumed that $\omega(\mathcal{C}) \geq 8k$.

We now introduce the concept of a *heavy neighbour* of a vertex $s \in S$ in $\mathcal{C}$, which is a key element for explaining the forthcoming reduction rules.

Definition 22. *A clique $C \in \mathcal{C}$ is called a heavy neighbour of $s \in S$ if s is adjacent to at least* $\max\{|C| - 4k, k + 1\}$ *vertices in C.*

The next reduction rule helps to show that each vertex s has a unique heavy neighbour in $\mathcal{C}$.

Reduction Rule UCVD 5. *If $s \in S$ has no heavy neighbour, remove s from G, and decrement the parameter k by 1. The new instance is $(G - s, k - 1)$.*

Lemma 10. *Reduction Rule UCVD 5 is safe.*

Proof. For the sake of contradiction, let us assume that $s \in S$ has no heavy neighbour and does not belong to some solution X of size at most k. It means s must belong to some clique in $G - X$. Since s has no heavy neighbour in $\mathcal{C}$, s is adjacent to at most $\max\{|C| - 4k, k + 1\} - 1$ vertices in some $C \in \mathcal{C}$. Additionally, s can have at most $3k - 1$ neighbours in S. Now consider two cases:

1. If $\max\{|C| - 4k, k+1\} - 1 = |C| - 4k - 1$, then s can be part of a clique of size at most $(|C| - 4k - 1) + (3k - 1) + 1 = |C| - k - 1$. This contradicts the fact that $G - X$ is a collection of r-cliques, where $r \geq |C| - k$.

2. If $\max\{|C| - 4k, k+1\} - 1 = k$, then s can be part of a clique of size at most $k + (3k-1) + 1 = 4k$. This contradicts the fact that $G - X$ is a collection of r-cliques, where $r \geq 7k$.

Therefore, s must be part of every solution of size at most k. □

Lemma 11. *After applying the Reduction Rules 1 and 5 exhaustively, each s has exactly one heavy neighbour.*

Proof. The condition 2 in the Reduction Rule 1 makes sure that s has at most one heavy neighbour. On the other hand, the Reduction Rule 5 makes sure that s has at least one heavy neighbour. Therefore, it shows that each s has exactly one neighbour. □

Let $S_C \subseteq S$ denotes the set of vertices in S whose heavy neighbour is C. For each $C \in \mathcal{C}$, we define

$$\Gamma(C) := \bigcup_{s \in S_C} \Big(C \setminus N(s)\Big) \cup \bigcup_{s \in S \setminus S_C} \Big(N(s) \cap C\Big) \quad \text{and} \quad \Gamma(\mathcal{C}) = \bigcup_{C \in \mathcal{C}} \Gamma(C).$$

Next, we show that the vertices of $C \setminus \Gamma(C)$ are true twins.

Lemma 12. *For each $C \in \mathcal{C}$, every pair of adjacent vertices x and y in $C \setminus \Gamma(C)$ are true twins, that is, $N[x] = N[y]$.*

Proof. By the definition of $\Gamma(C)$, note that for any vertex $v \in C \setminus \Gamma(C)$, we have $N(v) \cap S = S_C$. Recall that for the vertices in S_C, C is their heavy neighbour. Since $v \in C$ and C is a clique, v is adjacent to all other vertices of C. Therefore, for every $v \in C \setminus \Gamma(C)$, we have $N[v] = C \cup S_C$. □

Reduction Rule UCVD 6. *If $\min_{C \in \mathcal{C}}\{|C \setminus \Gamma(C)|\} > k+1$, then arbitrarily delete exactly one vertex from each $C \setminus \Gamma(C)$ for every $C \in \mathcal{C}$. The new instance is $(G - W, k)$, where W is the set of vertices deleted from G.*

Lemma 13. *Reduction Rule UCVD 6 is safe.*

Proof. Let $I' = (G', k)$ be an instance of UCVD obtained from $I = (G, k)$ by exhaustively applying Reduction Rule 6. We will show that I is a yes-instance if and only if I' is a yes-instance.

Let $X \subseteq V(G)$ be a solution of size at most k for the instance I. We will construct a solution X' of the same size as X for the instance I'. We know $G - X$ is a collection of equal-sized cliques $Q_1, Q_2, \ldots, Q_r$. We observe that there is no clique Q_i such that $Q_i \subseteq S$. This is true because otherwise $|Q_i| \leq 3k$ as $|S| \leq 3k$, and we have already observed that the size of cliques must be at least $7k$. We construct X' from X as follows: if $u \in X \cap (C \setminus \Gamma(C))$ for some $C \in \mathcal{C}$ and u is deleted during the execution of Reduction Rule 6, then remove u from X and include u's true twin $v \notin X$. We have to show that $G' - X'$ is a collection of equal-sized cliques. It is easy to see that $G' - X'$ is a collection of cliques $Q'_i = Q_i \setminus \{u\}$ where $u \in Q_i$ is a vertex deleted by Reduction Rule 6. As $|Q_i| = |Q_j|$, we get $|Q'_i| = |Q'_j|$ for all $i \neq j$. This shows that I' is a yes-instance.

In the other direction, let us assume that X' is a solution of size at most k for the instance I'. Let us denote the equal-sized cliques in $G' - X'$ by $Q'_1, Q'_2, \ldots, Q'_r$. We claim that $X := X'$ is a solution for the instance I. Note that $G - X$ is again a collection of cliques $Q_1, Q_2, \ldots, Q_r$ where $Q_i = Q'_i \cup \{u\}$ where u is a vertex deleted by Reduction Rule 6. Clearly, we have $|Q_i| = |Q_j|$ as we know that $|Q'_i| = |Q'_j|$ for all $i \neq j$. □

Lemma 14. *After the exhaustive application of the above reduction rules, we have* $\max_{C \in \mathcal{C}} |C| \leq \mathcal{O}(k^2)$.

Proof. We begin by providing an upper bound on $|\Gamma(\mathcal{C})|$.

Claim 3.1.1. *We have* $|\Gamma(\mathcal{C})| = \mathcal{O}(k^2)$.

Proof. Let us assume that C_s denotes the unique heavy neighbour of s. We have

$$\begin{aligned}\Gamma(\mathcal{C}) &= \bigcup_{C\in\mathcal{C}}\left[\bigcup_{s\in S_C}\Big(C\setminus N(s)\Big)\cup\bigcup_{s\in S\setminus S_C}\Big(N(s)\cap C\Big)\right]\\ &= \bigcup_{s\in S}\left[\Big(C_s\setminus N(s)\Big)\cup\bigcup_{C\in\mathcal{C}\setminus C_s}\Big(N(s)\cap C\Big)\right]\end{aligned}$$

Note that by Reduction Rule 1, for any fixed $s\in S$, we have

$$\left|\bigcup_{C\in\mathcal{C}\setminus C_s}\Big(N(s)\cap C\Big)\right| < (k+1).$$

Also, for each $s\in S$, we have

$$\begin{aligned}\big|C_s\setminus N(s)\big| &\leq |C|-(|C|-4k) = 4k \text{ if } \max\{|C|-4k, k+1\} = |C|-4k\\ &\leq |C|-(k+1) \leq 4k \text{ if } \max\{|C|-4k, k+1\} = k+1\end{aligned}$$

As there are at most $3k$ vertices in S, we get that $|\Gamma(\mathcal{C})|\leq 3k[k+4k] = 15k^2 = \mathcal{O}(k^2)$. ▷

After exhaustively applying Reduction Rule 6, we know that there exists a clique, say $C_q\in\mathcal{C}$, such that $|C_q|-|\Gamma(C_q)|\leq k+1$. Note that $\max_{C\in\mathcal{C}}|C|-|C_q|\leq 4k$ due to Reduction Rule UCVD 4. Therefore, we get $\max_{C\in\mathcal{C}}|C|-|\Gamma(C_q)|\leq 5k+1$. Since $|\Gamma(C_q)| = \mathcal{O}(k^2)$, we get $\max_{C\in\mathcal{C}}|C|\leq\mathcal{O}(k^2)$. □

Note that

$$V(G) = S\cup\bigcup_{C\in\mathcal{C}}C = S\cup\bigcup_{C\in\mathcal{C}}\big(\Gamma(C)\cup(C\setminus\Gamma(C))\big) = S\cup\Gamma(\mathcal{C})\cup\bigcup_{C\in\mathcal{C}}(C\setminus\Gamma(C))$$

To get a cubic kernel, it is enough to show that the size of the set $\bigcup_{C\in\mathcal{C}}(C\setminus\Gamma(C))$ is bounded by a cubic function of the parameter. To see this, we partition the set of cliques in $\mathcal{C}$ into two parts: $\mathcal{H} = \{C\in\mathcal{C}\mid C \text{ is a heavy neighbour of some } s\in S\}$ and $\mathcal{L} = \mathcal{C}\setminus\mathcal{H}$. As we have seen, every vertex $s\in S$ has exactly one heavy neighbour, which implies that $|\mathcal{H}|\leq 3k$. Therefore, we get that $\bigcup_{C\in\mathcal{H}}|(C\setminus\Gamma(C))| = \mathcal{O}(k^3)$ as $\max_{C\in\mathcal{C}}|C|\leq\mathcal{O}(k^2)$. Now, let us focus on the set $\mathcal{L}$. Let us denote $\mathcal{L}' = \{C\in\mathcal{L}\mid |C\setminus\Gamma(C)|\geq k+1\}$ and $\mathcal{L}'' = \mathcal{L}\setminus\mathcal{L}'$. It is clear

that $\bigcup_{C \in \mathcal{L}''} |(C \setminus \Gamma(C))| \leq \mathcal{O}(k^3)$ as $|\mathcal{L}''| \leq |\mathcal{C}| \leq \mathcal{O}(k^2)$. We can assume that $|\mathcal{L}'| \geq 2k+1$; otherwise, we achieve the required cubic kernel.

Lemma 15. *If $|\mathcal{L}'| \geq 2k+1$ then in polynomial time we can determine the size c of equal-sized cliques obtained after deleting the vertices in a solution, assuming the input is a yes-instance.*

Proof. Let us denote $\mathcal{L}' = \{C'_1, C'_2, \ldots, C'_l\}$, where $l \geq 2k+1$. Let X be a solution of size at most k of the input instance. As $|C'_i \setminus \Gamma(C'_i)| \geq k+1$, at least one vertex will survive from each set $C'_i \setminus \Gamma(C'_i)$ in $G - X$. Because the vertices in $C'_i \setminus \Gamma(C'_i)$ do not have any neighbours in S, the deletion of at most k vertices can change the degree of vertices in at most k sets $C'_i \setminus \Gamma(C'_i)$. This implies that vertices in all but at most k distinct sets $C'_i \setminus \Gamma(C'_i)$ must have same degree. If this is not the case, we can return a no-instance. If it is the case, then let us say the degree is d. One can easily see that $c = d+1$. It is important to note that having $|\mathcal{L}'| \geq 2k+1$ is necessary; otherwise, identifying d is not possible. □

The following set of reduction rules are only applicable when $|\mathcal{L}'| \geq 2k+1$.

Reduction Rule UCVD 7. *If there is a vertex v such that $d(v) < d$, then add it to the solution. The new instance is $(G - v, k-1)$.*

Note that the degree of every vertex in $G - X$ must be equal to d. Since $d(v) < d$ and we cannot increase the degree of v through vertex deletion, v must be added to the solution. Due to Reduction Rule 7, we can assume that all the cliques in $\mathcal{L}'$ have size at least c.

Reduction Rule UCVD 8. *If there is a clique $C' \in \mathcal{L}'$ such that $|C'| = c$, then delete all the vertices in $N(C')$ from the graph G. The new instance is $(G - N(C'), k - |N(C')|)$.*

Lemma 16. *Reduction Rule 8 is safe.*

Proof. Note that at least one vertex $u \in C' \setminus \Gamma(C')$ must survive after deleting the vertices in a solution. This vertex must be part of a clique of size c. Therefore, the clique C' must appear unchanged in the resulting graph. Hence, we must delete all the neighbours of C'. □

Let us denote by $\mathcal{L}'_{iso,c}$ the set of isolated cliques of size c in $\mathcal{L}'$ obtained after applying Reduction Rule 8. Similarly, denote by $\mathcal{L}'_{>c}$ the set of cliques of size more than c in $\mathcal{L}'$. Note that, we have $\mathcal{L}' = \mathcal{L}'_{iso,c} \cup \mathcal{L}'_{>c}$.

Observation 3.1.2. *Due to application of Reduction Rule 2, we get $|\mathcal{L}'_{iso,c}| \leq k+1$.*

Reduction Rule UCVD 9. *If $|\mathcal{L}'_{>c}| > k$, then conclude that we are dealing with a no-instance.*

If $|\mathcal{L}'_{>c}| > k$, it implies we need to delete more than k vertices to ensure that each clique in $\mathcal{L}'_{>c}$ becomes a clique of size c in the resulting graph. However, since we are constrained to finding a solution of size at most k, achieving this is not feasible. Therefore, $|\mathcal{L}'_{>c}| \leq k$.

Observation 3.1.3. *After applying Reduction Rules 7, 8 and 9, the size of $\mathcal{L}'$ is bounded by $2k+1$.*

We know that $\mathcal{L}' = \mathcal{L}'_{iso,c} \cup \mathcal{L}'_{>c}$. Therefore, $|\mathcal{L}'| \leq |\mathcal{L}'_{iso,c}| + |\mathcal{L}'_{>c}| \leq (k+1)+k \leq 2k+1$.

Clearly, due to Observation 3.1.3, we have $\bigcup_{C \in \mathcal{L}'} |C \setminus \Gamma(C)| \leq \mathcal{O}(k^3)$. This completes the proof of Theorem 5.

Theorem 17. *The* UCVD *problem parameterized by solution size k can be solved in $2^k \cdot n^{\mathcal{O}(1)}$ time.*

Proof. We utilize an algorithm from [108] to find a cluster vertex deletion set X of size at most k in $\mathcal{O}^*(1.811^k)$ time. If this algorithm determines that no such set exists, conclude that we are dealing with a no-instance. Otherwise, this algorithm returns a cluster vertex deletion set X of size at most k. To decide whether G contains a uniform cluster vertex deletion set S of size at most k, we proceed as follows:

- We guess the intersection of S with X, that is, we guess the set $X_{in} = X \cap S$, delete X_{in} from G and reduce parameter k by $|X_{in}|$.
- For each guess of X_{in}, we set $X_{out} = X \setminus X_{in}$ and solve Disjoint UCVD on the instance $(G - X_{in}, X_{out}, k - |X_{in}|)$.

For each guess X_{in}, we try to find a uniform cluster vertex deletion set S' in $G - X_{in}$ of size at most $k - |X_{in}|$ that is disjoint from X_{out}. This problem is called Disjoint UCVD. If for some guess X_{in}, we find a uniform cluster vertex deletion set S' in $G - X_{in}$ of size at most $k - |X_{in}|$ that is disjoint from X_{out}, then we output $S = X_{in} \cup S'$. Otherwise, we conclude that the given instance of the Disjoint UCVD problem is a no-instance. The

number of all guesses is bounded by 2^k. Therefore, to obtain an FPT algorithm for UCVD, it is sufficient to solve DISJOINT UCVD in polynomial time.

An algorithm for DISJOINT UCVD: Let $(G - X_{in}, X_{out}, k)$ be an instance of DISJOINT UCVD, and let $H = G - X$, where $X = X_{in} \cup X_{out}$. We denote $G' = G - X_{in}$.

Reduction Rule Disjoint-UCVD 1. *If $G[X_{out}]$ is not a disjoint union of cliques or $G - X$ is not a disjoint union of cliques, then return that (G', X_{out}, k) is a no-instance.*

Reduction Rule Disjoint-UCVD 2. *If there is a vertex v in $G - X$ which has neighbours in two distinct cliques in $G[X_{out}]$, then delete v from G and decrement the parameter k by 1. The new instance is $(G' - \{v\}, X_{out}, k - 1)$.*

The previous reduction rule is correct because we will get an induced P_3 where the only vertex that we are allowed to delete is v. Therefore, we must add v to the solution.

So from now onwards, we assume that $G[X_{out}]$ is a disjoint union of cliques and $H = G - X$ is a disjoint union of cliques. We also guess the exact size of the solution as k'. The possible guesses for k' range from 1 to k. This will be helpful in the final part of the algorithm. We will solve this problem by guessing the size c of cliques in the resulting graph. At the end, we will compare the solution sizes for each c and return the minimum one. The possible guesses for c ranges from 1 to n.

Reduction Rule Disjoint-UCVD 3. *If there is a clique of size exactly c in $G[X_{out}]$ then delete $N(C)$ and add all the vertices of the set $N(C)$ to the solution. The new instance is $(G' - N[C], X_{out}, k - |N(C)|)$.*

Observation 3.1.4. *If there is a clique C of size larger than c in $G[X_{out}]$, then we can discard such a guess for c.*

Observation 3.1.5. *Every clique in $G[X_{out}]$ of size less than c must utilize vertices from exactly one clique in $G - X$ to form a clique of size c.*

Due to Observation 3.1.5, for every clique C of size less than c in $G[X_{out}]$, there is a unique clique in $G - X$ that helps to form a clique of size c containing vertices from C.

We now construct a bipartite graph with bipartition (A, B) in the following way. For a given guess (k', c), observe that the number of equal-sized cliques in the resulting graph is $p = \frac{|V(G')|-k'}{c}$. We need a total of p vertices in A. Suppose the number of cliques in $G[X_{out}]$ is p_1. We add a vertex $a \in A$ corresponding to each clique in $G[X_{out}]$. If $p_1 < p$, we add $p - p_1$ dummy vertices in A to ensure the number of vertices in A equals p. Similarly, we add a vertex $b \in B$ corresponding to each clique in $G - X$. Due to Reduction Rule Disjoint-UCVD 3 and Observation 3.1.4, it follows that every clique in $G[X_{out}]$ has size less than c. We add an edge between $a \in A$ and $b \in B$ if the clique corresponding to a can be transformed into a clique of size c using some vertices from the clique corresponding to b. Additionally, some cliques in $G - X$ can be transformed into cliques of size c by deleting the necessary number of vertices from the respective cliques. We make $b \in B$ adjacent to all dummy vertices in A if the clique corresponding to b can be transformed into a clique of size c by deleting the necessary number of vertices.

Lemma 18. *There exists an A-saturated matching if and only if there is a set of p disjoint cliques, each of size exactly c, obtained by deleting exactly k' vertices from $G - X$, such that the graph induced by these cliques forms a uniform cluster graph.*

Proof. In the forward direction, suppose there exists an A-saturated matching. We can identify p cliques in the resulting graph as follows:

- If $(a, b) \in M$ and a is not a dummy vertex, the clique corresponding to $a \in A$ will be transformed into a clique of size c using some vertices from the clique corresponding to $b \in B$. - If $(a, b) \in M$ and a is a dummy vertex, the clique corresponding to $b \in B$ will be adjusted to a clique of size c by deleting the necessary vertices from that clique.

Thus, we can identify the exact cliques in the final graph. Since $p \cdot c = |V(G')| - k'$, we know exactly which k' vertices need to be deleted to form these cliques of size c.

In the backward direction, assume there is a set of p disjoint cliques, each of size exactly c, obtained by deleting exactly k' vertices from $G - X$, and that the induced subgraph on these cliques forms a uniform cluster graph. By our construction, we can obtain an edge in the bipartite graph (A, B) corresponding to each clique, forming a matching. Since there are p cliques, this matching will be A-saturated. □

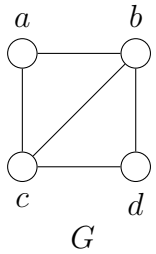
We now try to find an A-saturated matching in this bipartite graph. This can be done in

polynomial time. Due to the previous claim, if an A-saturated matching does not exist, we discard such a guess (k', c). Otherwise, we find an A-saturated matching M of size p and return a yes instance. This finishes the proof of Theorem 17.

3.2 Kernelization algorithm for UCVS parameterized by solution size

Suppose G is a graph with an isolated vertex. If (G, k) is a yes-instance, then the size of the equal-sized cliques in the resulting graph must be 1. Therefore, if (G, k) is yes-instance, G itself must be a disjoint union of isolated vertices. This means,we can solve (G, k) in polynomial time if G contains an isolated vertex. Hence, in this section, we consider only graphs that do not contain any isolated vertices.

In the example below, we demonstrate that inclusive vertex splitting is a stronger operation than exclusive vertex splitting. In the following graph, we can obtain a disjoint union of triangles by performing just two inclusive vertex splitting operations. However, if we allow only exclusive vertex splitting, then at least four vertex splitting operations are required to transform G into a uniform cluster graph.



G

In this section, we prove that UCIVS and UCEVS are NP-hard. Our proof is based on a reduction from a problem known as the K_d-Edge Partition problem.

Definition 23. *A graph $G = (V, E)$ is said to admit a K_d-edge partition if there exists a partition of the edge set E into subsets $\mathcal{P} = \{P_1, P_2, \ldots, P_l\}$, such that the subgraph $G[P_i]$ induced by the edges in each P_i is isomorphic to a clique of size d.*

The following problem was shown to be NP-complete by Ian Holyer [78] for every $d \geq 3$.

K_d-Edge Partition
Input: An undirected graph G.
Question: Does there exist a K_d-edge partition $\mathcal{P}$ of G?

To make the explanation easier, we introduce the notion of the cost of a partition.

Definition 24. *Let $\mathcal{P}$ be a K_d-edge partition of a graph $G = (V, E)$. The cost of $\mathcal{P}$, denoted by $\texttt{cost}(\mathcal{P})$, is defined as:*

$$\texttt{cost}(\mathcal{P}) = \sum_{v \in V} \max(0, \text{freq}_{\mathcal{P}}(v) - 1),$$

where $\text{freq}_{\mathcal{P}}(v) := |\{P \in \mathcal{P} \mid v \in V(G[P])\}|$.

Weighted K_d-Edge Partition
Input: An undirected graph $G = (V, E)$, and a positive integer k.
Question: Does there exist a K_d-edge partition $\mathcal{P}$ of G such that $\texttt{cost}(\mathcal{P}) \leq k$?

We present a few lemmas to simplify and clarify the NP-hardness proof.

Lemma 19. *Let $G = (V, E)$ be a graph with no isolated vertices, and let $\mathcal{P}$ be a K_d-edge partition of G with $\texttt{cost}_G(\mathcal{P}) = k$. Then, there exists a vertex $u \in V$ such that by performing an exclusive vertex splitting of u in G, we obtain a new graph $G_1 = (V_1, E_1)$ satisfying:*

1. *G_1 admits a K_d-edge partition, say $\mathcal{P}'$,*
2. *$\texttt{cost}_{G_1}(\mathcal{P}') = k - 1$.*

Proof. If G is not already a disjoint union of d-sized cliques, without loss of generality there must exist P_1 and P_2 such that $V(G[P_1]) \cap V(G[P_2]) \neq \emptyset$. In this case, let $u \in V(G[P_1]) \cap V(G[P_2])$. We define G' as the graph that is obtained when u is split into the two vertices u_1 and u_2 such that

$$N_{G_1}(u_1) := N_G(u) \cap V(G[P_1]),$$
$$N_{G_1}(u_2) := N_G(u) \setminus V(G[P_1]).$$

Furthermore, we obtain a K_d-edge partition of G_1 as $\mathcal{P} = \{P'_1, P'_2, \ldots, P'_l\}$ such that

$$P'_i := \begin{cases} \{\{x,y\} \in P_1 \mid u \notin \{x,y\}\} \cup \{\{u_1, v\} \mid \{u,v\} \in P_1\} & \text{if } i = 1 \\ \{\{x,y\} \in P_i \mid u \notin \{x,y\}\} \cup \{\{u_2, v\} \mid \{u,v\} \in P_i\} & \text{if } 1 < i \leq l \end{cases}$$

Note that $\text{freq}_{\mathcal{P}}(u) = \text{freq}_{\mathcal{P}'}(u_1) + \text{freq}_{\mathcal{P}'}(u_2)$. In particular, we have $\text{freq}_{\mathcal{P}'}(u_1) = 1$ and $\text{freq}_{\mathcal{P}'}(u_2) = \text{freq}_{\mathcal{P}'}(u) - 1$. Therefore, we get $\text{cost}_{G_1}(\mathcal{P}') = k - 1$. □

Lemma 20. *The instance (G, k) is a yes-instance of* UCEVS *if and only if it is a yes-instance of* Weighted K_d-Edge Partition *for some d.*

Proof. Let $I = (G, k)$ be a yes-instance, that is, there exists a sequence of at most k exclusive vertex splittings in G such that the resulting graph G' is a cluster graph consisting of disjoint cliques, each of size d. Since we are performing exclusive vertex splitting, for every edge $u'v' \in E(G')$, there exists a unique edge uv in $E(G)$ such that either $u' = u$ or u' is a descendant of u, and similarly $v' = v$ or v' is a descendant of v. Therefore, each K_d-cliques in G' corresponds uniquely to a K_d-clique in G. This yields a K_d-edge partition of G. Since at most k vertex splittings are performed, the total number of vertices in G' is at most $|V(G)| + k$. Hence, (G, k) is a yes-instance of the Weighted K_d-Edge Partition problem.

For the reverse direction, suppose $\mathcal{P}$ is a K_d-edge partition of G with $\texttt{cost}(\mathcal{P}) \leq k$. By applying Lemma 19 iteratively at most k times, there exists a sequence of exclusive vertex splittings that transforms G into a disjoint union K_d-cliques. Therefore, (G, k) is also a yes-instance of the original problem. □

Theorem 21. Uniform Cluster Inclusive Vertex Splitting *is NP-complete.*

Proof. It is clear that the problem is in NP. To show that the problem is NP-hard, we provide a reduction from K_3-Edge Partition. The problem asks whether a given graph can be edge-partitioned into subgraphs isomorphic to the complete graph K_3 (triangles). Ian Holyer proved that this problem is NP-hard [78]. Given an instance $I = G(V, E)$ of K_3-Edge Partition, we construct an instance $I' = (G', k')$ of Uniform Cluster Inclusive Vertex Splitting. To construct the graph G', we start with G and add $m - n + 1$ disjoint

triangles, where $|V(G)| = n$ and $|E(G)| = m$. We set $k' = m - n$. Next, we prove that the instances I and I' are equivalent.

Assume that I is a yes-instance, that is, there exists a partition $\mathcal{P}$ of the edges of G such that the subgraph induced by each part is isomorphic to K_3. We denote this partition as

$$\mathcal{P} = \Big\{P_1, P_2, \ldots, P_{\frac{m}{3}} \mid G[P_i] \cong K_3, \forall i \in \{1, 2, \ldots, \frac{m}{3}\}\Big\}.$$

If such a partition exists then each vertex in G must have an even degree. For each vertex $v \in V(G)$, the number of triangles containing v is $l = \frac{d(v)}{2}$, since each triangle contributes 2 to the degree of each of its vertices. Therefore, the number of vertex splits required is

$$\texttt{cost}(\mathcal{P}) = \sum_{v \in V(G)} \left(\frac{d(v)}{2} - 1\right) = m - n.$$

Now by repeatedly applying Lemma 19 with $K_d = K_3$, we can perform vertex splittings to transform G into a disjoint union of triangles. Note that the total number of vertex splittings required is exactly $m - n$.

Now assume that I' is a yes-instance, that is, there exists a sequence of at most $m - n$ vertex splittings in G' that results in a uniform cluster graph G''. We first argue that G'' must be a disjoint union of cliques of size 3. Since G' includes $m - n + 1$ disjoint triangles and only $m - n$ vertex splittings are allowed, at least one of these triangles must remain unaltered. Therefore G'' contains at least one disjoint triangle. As G'' is a uniform cluster graph, and it contains at least one triangle, all its components must be isomorphic to K_3. We can assume without loss of generality that no vertex in $V(G') \setminus V(G)$ is split.

Claim 3.2.1. *For every edge $u''v'' \in E(G'')$, there exists a unique edge uv in $E(G')$ where either $u'' = u$ or u'' is a descendant of vertex u and similarly $v'' = v$ or v'' is a descendant of vertex v.*

Proof. Note that after $m - n$ vertex splittings, the graph G'' can contain at most $n + 3(m - n + 1) + (m - n) = m + 3(m - n + 1)$ many vertices. Since G'' is disjoint union of cliques of size 3, $|E(G'')| = \frac{3|V(G'')|}{3} = |V(G'')| \leq m + 3(m - n + 1) = |E(G')|$. Since G'' is obtained by vertex splitting operations on G', we must have $|E(G'')| \geq |E(G')|$. This implies that $|E(G'')| = |E(G')|$. Therefore, for every edge $u''v'' \in E(G'')$, there exists an unique edge uv in $E(G')$. ▷

The above claim shows that, for every triangle in G'', there exists a unique corresponding triangle in G'. As $|E(G'')| = |E(G')|$, this gives a partition of edges of G', where each part induces a K_3. Given such a partition of G', one can easily obtain a corresponding partition of the edges of G as well. This completes the proof of Theorem 21. □

Note that the above proof also applies to exclusive vertex splitting, as all the vertex splittings performed in the backward direction of the proof are exclusive.

Corollary 3.2.1. Uniform Cluster Exclusive Vertex Splitting *is NP-complete.*

3.2.1 A Kernelization Algorithm for UCEVS

Now, we will present a linear vertex kernel for the UCEVS.

Theorem 22. Uniform Cluster Exclusive Vertex Splitting *parameterized by solution size admits a kernel with at most* $4k$ *vertices.*

Proof. Let $G_0, G_1, G_2, ...G_l$ be a splitting sequence of length l for G. Let $S = \{v_1, v_2, \ldots, v_l\}$ denote the set of vertices that were split to obtain G_l from $G_0 = G$. Observe that for every vertex $x \in V \setminus S$, the degree of x remains unchanged throughout the splitting sequence. That is, $d_G(x) = d_{G_l}(x) = d$ for some fixed integer d. We assume that G has at least $2k + 1$ vertices. Otherwise, the graph itself forms a kernel of size at most $2k$ and we are done. If the input instance is a yes-instance, then all but at most k vertices must have the same degree in the original graph G. Therefore, we first check whether all but at most k vertices in G have the same degree. If not, we can safely conclude that the instance is a no-instance. If the input instance satisfies this condition, we can compute the common degree d in linear time. Assuming that the input is a yes-instance, let c be the size of equal-sized cliques in G_l. It follows that each clique must be of size $d + 1$.

Reduction Rule UCEVS 1. *If any of the following conditions hold, we conclude that the instance is a no-instance:*

1. $|\{v \in V(G) \mid d_G(v) > d\}| > k$,
2. *There exists a vertex* $v \in V(G)$ *such that* $d_G(v) < d$,

3. *There exists a vertex $v \in V(G)$ such that $d_G(v) = d$ and $G[N[v]]$ is not a clique.*

Lemma 23. *Reduction Rule 1 is correct.*

Proof. We justify the correctness of each condition in Reduction Rule 1 as follows:

1. **Condition 1**: If $|\{v \in V(G) \mid d_G(v) > d\}| > k$, then there are at least $k+1$ vertices whose frequency must be at least two in any K_{d+1}-edge partition. This implies that the cost of any such partition must exceed k. Therefore, the instance must be a no-instance.

2. **Condition 2**: If there exists a vertex $v \in V(G)$ such that $d_G(v) < d$, then clearly $G[N[v]]$ does not contain a clique of size $d+1$. This means that v cannot be part of any clique in a K_{d+1}-edge partition. Therefore, the instance is a no-instance.

3. **Condition 3**: If there exists a vertex $v \in V(G)$ such that $d_G(v) = d$ and $G[N[v]]$ is not a clique, then $G[N[v]]$ does not contain a clique of size $d+1$. This means that v cannot be part of any clique in a K_{d+1}-edge partition. Therefore, the instance is a no-instance.

Since all three conditions correctly identify invalid instances, the reduction rule is sound. □

We make two cases based on the value of d.

Case 1. Let $d > 2k$.

In this case, we provide a polynomial-time algorithm to solve the problem. Specifically, due to Lemma 20, we determine whether there exists a weighted K_{d+1}-edge partition of G with cost at most k.

Algorithm 3.2.1: Weighted K_{d+1}-Edge Partition

Require: A graph $G = (V, E)$ and an integer k.

Ensure: A weighted K_{d+1}-edge partition $\mathcal{P}$ of cost at most k, or conclude that no such partition exists.

1: Initialize an empty partition: $\mathcal{P} \leftarrow \emptyset$
2: **while** $G \neq \emptyset$ **do**
3: Find a vertex $v \in V(G)$ such that $d_G(v) = d$
4: **if** no such vertex exists **then**
5: **return** "No instance."
6: **end if**
7: Let $C = G[N[v]]$, the subgraph induced by the closed neighborhood of v
8: **if** C is not a clique **then**
9: **return** "No instance."
10: **end if**
11: Add the edges of C to $\mathcal{P}$:

$$\mathcal{P} \leftarrow \mathcal{P} \cup \{E(C)\}$$

12: **Check Consistency of $\mathcal{P}$:**

- If any two elements $P_i, P_j \in \mathcal{P}$ share an edge (i.e., $P_i \cap P_j \neq \emptyset$), return "No instance."
- Compute $\text{cost}(\mathcal{P}) = \max(0, \text{freq}_{\mathcal{P}}(v) - 1)$. If $\text{cost}(\mathcal{P}) > k$, return "No instance."

13: **Update G and k:**

$$V(G) \leftarrow V(G) \setminus \{u \in N[v] \mid d_G(u) = d\},$$
$$k \leftarrow k - \text{cost}(\mathcal{P})$$

14: **end while**
15: **return** $\mathcal{P}$

Lemma 24. *The Weighted K_{d+1}-Edge Partition Algorithm (Algorithm 3.2.1) correctly outputs a weighted K_{d+1}-edge partition of cost at most k, or conclude that no such partition exists.*

Proof. The correctness of the algorithm is established as follows:

1. **Initial If Conditions**: The first two `if` conditions (lines 4 and 7 of the algorithm)

directly follow from **Reduction Rule 1**:

- If no vertex v exists with $d_G(v) = d$, then it is impossible to construct a valid K_{d+1}-edge partition, as per **Condition 1** of the reduction rule.
- If $G[N[v]]$ is not a clique for a vertex v with $d_G(v) = d$, then v cannot belong to any K_{d+1}-clique, as per **Condition 3** of the reduction rule.

2. **Partition Update**: When a vertex v with $d_G(v) = d$ is found, all edges adjacent to v must belong to the same part of the partition $\mathcal{P}$. This is guaranteed by the definition of a K_{d+1}-edge partition, which requires that $G[N[v]]$ form a clique. Therefore, adding $E(G[N[v]])$ to $\mathcal{P}$ is a valid and safe update.

3. **Consistency Checks**:

- The **first consistency condition** holds because $\mathcal{P}$ is explicitly defined as a partition of the edge set, ensuring that no two parts overlap.
- The **second consistency condition** holds due to **Lemma 20**, which guarantees that cost of the partition $\text{cost}(\mathcal{P})$ does not exceed the allowed budget k.

4. **Graph Updates**: Since all edges adjacent to the vertices in the set $\{u \in N[v] \mid d_G(u) = d\}$ have been included in $\mathcal{P}$, it is safe to delete these vertices and their incident edges from G. Additionally, the budget k is correctly reduced by the amount $\text{cost}(\mathcal{P})$, reflecting the edges already accounted for in the partition.

5. **Termination**: The algorithm terminates when G becomes empty, at which point all edges have been successfully partitioned into K_{d+1}-cliques. If any condition fails during execution, the algorithm correctly concludes a no-instance. Since $d \geq 2k$, every vertex of degree greater than d is adjacent to a vertex of degree d otherwise Reduction Rule UCIVS 1 could have been applied. Note that after each iteration, if G is not empty, we will either have a vertex of degree d or all vertices have a degree not equal to d. In the latter case, we return no-instance by Reduction Rule 1 and in the former case, since we have a vertex with degree d, we execute the line 3 of the algorithm. This means that either G becomes empty or any condition fails during execution.

Thus, the algorithm is correct. □

Case 2. Let us assume $d \leq 2k$.

There is a subtle difference between the two cases. In Case 1, after each iteration of updating G, one of two outcomes occurs: either the graph G becomes empty, or $|V(G)| \geq 2k+1$. In the former case, we can output the partition directly. In the latter case, Reduction Rule 1 guarantees that at least $k+1$ vertices (and all but k) have degree d. This ensures that if the reduced instance is a yes-instance, the equal-sized cliques in the final graph must be of size $d+1$, which is consistent with the original instance.

In Case 2, we must ensure that after each update of G, the graph either becomes empty or it contains at least $2k+1$ vertices. Here, Reduction Rule 1 similarly ensures that at least $k+1$ vertices (and all but k) have degree d. Since $d \leq 2k$, the graph must have at least $4k+1$ vertices to satisfy these conditions. Therefore, we apply the Weighted K_{d+1}-Edge Partition Algorithm (Algorithm 3.2.1) only if $|V(G)| > 4k$.

Reduction Rule UCEVS 2. *If $|V(G)| > 4k$, then run the Weighted K_{d+1}-Edge Partition Algorithm (Algorithm 3.2.1) on the graph G, but terminate the algorithm if the size of the vertex set becomes $|V(G)| \leq 4k$.*

The correctness of the Reduction Rule 2 follows from correctness of Weighted K_{d+1}-Edge Partition Algorithm (Algorithm 3.2.1) as established by Lemma 24. This completes the proof of Theorem 22. □

3.2.2 A Kernelization Algorithm for UCIVS

We will present a linear kernel for the UCIVS. We first provide some lemmas given by Firbas et.al [61].

Lemma 25. *[61] Let G be a non-empty graph and G' be obtained from G by splitting some vertex $w \in V(G)$ into $w_1, w_2 \in V(G')$; furthermore let $v_1, v_2 \in V(G)$. If $v_1v_2 \notin E(G)$, then for all descendants $v_1', v_2' \in V(G')$ of v_1 and v_2 respectively, it holds that $v_1'v_2' \notin E(G')$.*

Lemma 26. *[61] Let G be a non-empty graph and G' be obtained from G by splitting some vertex $w \in V(G)$ into $w_1, w_2 \in V(G')$; furthermore let $v_1, v_2 \in V(G)$. If $v_1v_2 \in E(G)$, then there are two descendants $v_1', v_2' \in V(G')$ of v_1 and v_2 respectively, such that $v_1'v_2' \in E(G')$.*

Note that in the previous subsection, we saw correlation between the problems UCEVS and K_{d+1}-Edge Partition. Here, we see a similar trend. We correlate the UCIVS and a problem called Sigma Uniform Clique Cover (SUCC).

Definition 25. A family $\mathcal{C}$ of cliques in G is called an *Sigma uniform clique cover* if

1. for each $e \in E$, there exists $C \in \mathcal{C}$ such that $e \in E(G[C])$ and
2. all the cliques in the family $\mathcal{C}$ have the same size.

Definition 26. *A Sigma uniform clique cover where cliques are of size s is called a Sigma s-clique cover.*

Definition 27. *Let $\mathcal{C}$ be a Sigma uniform clique cover of G. The weight of $\mathcal{C}$, denoted by $\mathtt{wgt}(\mathcal{C})$, is defined as:*

$$\mathtt{wgt}(\mathcal{C}) = \sum_{v \in V} \text{freq}_{\mathcal{C}}(v),$$

where $\text{freq}_{\mathcal{C}}(v) := |\{C \in \mathcal{C} \mid v \in V(G[C])\}|$.

A similar definition of weight of $\mathcal{C}$ is given in [61]. We also define a cost of $\mathcal{C}$. The definition of cost of $\mathcal{C}$ will be useful in the explanation of kernelization result.

Definition 28. *Let $\mathcal{C}$ be a Sigma uniform clique cover of G. The cost of $\mathcal{C}$, denoted by $cost(\mathcal{C})$, is defined as:*

$$cost(\mathcal{C}) = \sum_{v \in V} \max(0, \text{freq}_{\mathcal{C}}(v) - 1),$$

where $\text{freq}_{\mathcal{C}}(v) := |\{C \in \mathcal{C} \mid v \in V(G[C])\}|$.

The following two lemmas are closely related to lemmas proved in [61]. Lemmas given in [61] dealt with the problem Cluster Inclusive Vertex Splitting relating it to Sigma Clique Cover. Here, we see that the same idea works while working with the UCIVS as well by taking Sigma Uniform Clique Cover instead of Sigma Clique Cover.

Lemma 27. *Let $G = (V, E)$ be a graph and let $G' = (V', E')$ be obtained from G by splitting $u \in V$ into $v, w \in V'$. If $\mathcal{C}'$ is a Sigma uniform clique cover of G', then there exists a Sigma uniform clique cover $\mathcal{C}$ of G with $\mathtt{wgt}(\mathcal{C}) = \mathtt{wgt}(\mathcal{C}')$.*

Proof. Using

$$f(C') := \begin{cases} (C' \setminus \{v, w\}) \cup \{u\} & \text{if } C' \cap \{v, w\} \neq \emptyset \\ C' & \text{otherwise} \end{cases}$$

we define

$$\mathcal{C} := \{f(C') \mid C' \in \mathcal{C}'\}.$$

Note that f gives a bijection over the domain $\mathcal{C}'$. We claim that $\mathcal{C}$ satisfies the conditions of this lemma. First, we establish that $\mathcal{C}$ is a Sigma uniform clique cover of G. We note that the map f is the same as the map given in the proof of Lemma 4.1 of [61]. So, we can conclude that $C \in \mathcal{C}$ induce cliques in G and all edges of G are covered by $\mathcal{C}$. If $C' \cap \{v, w\} \neq \emptyset$, then since C' is a clique, it can not have both v and w, it means C' contains exactly one vertex from v and w. Therefore $|f(C')| = |C'|$ when $C' \cap \{v, w\} \neq \emptyset$. It is clear that $|f(C')| = |C'|$ when $C' \cap \{v, w\} = \emptyset$. Hence f ranging over $\mathcal{C}'$ does not change the cardinality of any image it maps, this observation says that all cliques in $\mathcal{C}$ are of the same size. It is also clear that $\texttt{wgt}(\mathcal{C}) = \texttt{wgt}(\mathcal{C}')$. □

Lemma 28. *Let $G = (V, E)$ be a graph with no isolated vertices and let $\mathcal{C}$ be a Sigma uniform clique cover of G with $\mathrm{wgt}(\mathcal{C}) \leq |V| + \alpha \in \mathbb{N}$. Then, either G is already a uniform cluster graph or there is $u \in V$ such that u can be split in G to obtain $G' = (V', E')$ satisfying*

1. *G' has a Sigma uniform clique cover $\mathcal{C}'$,*
2. *$\mathrm{wgt}(\mathcal{C}') \leq |V'| + \alpha - 1$,*
3. *G' does not contain isolated vertices.*

Proof. If G is not already a cluster graph, there must exist $C_1 \neq C_2 \in \mathcal{C}$ such that $C_1 \cap C_2 \neq \emptyset$. In this case, let $u \in C_1 \cap C_2$.

We define $G' = (V', E')$ as the graph that is obtained when u is split into the two vertices u_{in} and u_{out} obeying:

$$N_{G'}(u_{\text{in}}) := N_G(u) \cap C_1,$$
$$N_{G'}(u_{\text{out}}) := (N_G(u) \setminus C_1) \cup \{v \in N_G(u) \cap C_1 \mid \exists C \in \mathcal{C} \setminus \{C_1\} : u, v \in C\}.$$

Furthermore, using the map

$$f(C) := \begin{cases} (C \setminus \{u\}) \cup \{u_{\text{in}}\} & \text{if } C = C_1 \\ (C \setminus \{u\}) \cup \{u_{\text{out}}\} & \text{if } u \in C \wedge C \neq C_1 \\ C & \text{otherwise} \end{cases}$$

we can define

$$\mathcal{C}' := \{f(C) \mid C \in \mathcal{C}\}.$$

Note that f gives a bijection between $\mathcal{C}$ and $\mathcal{C}'$. Thus, $\mathcal{C}'$ is a sigma clique cover of G'. We note that all the construction that we did above is the same as the construction given in the proof of Lemma 4.2 of [61]. So we can conclude, that $\mathcal{C}'$ is a sigma clique cover of G' with $\text{wgt}(\mathcal{C}') \leq |V'| + \alpha - 1$ and G' does not contain any isolated vertices. The way we have defined f, it is clear that all cliques of $\mathcal{C}'$ are of the same size. □

Lemma 29. *An instance (G, k) is a yes-instance of* UCIVS *if and only if the corresponding instance $(G, |V| + k)$ is a yes-instance of* Sigma Uniform Clique Cover.

Proof. Let $G_0, \ldots, G_l$ be a sequence of graphs with $G_0 = G$ and $l \leq k$ such that each graph, except G_0, is obtained from its predecessor via a vertex split, and G_l is a uniform cluster graph. By identifying all connected components of G_l with their vertex sets, we can construct a Sigma uniform clique cover $\mathcal{C}_l$ of G_l with $\texttt{wgt}(\mathcal{C}_l) = |V(G_l)|$. Each split used in the construction of $G_0, \ldots, G_l$ introduces exactly one new vertex, therefore $|V(G_l)| = |V| + l$. Combining this with the fact that $l \leq k$, we derive $\texttt{wgt}(\mathcal{C}_l) \leq |V| + k$. Using the sequence $G_0, \ldots, G_l$ in reverse order, we iteratively apply Lemma 27 l times using $\mathcal{C}_l$ and G_l as base case and obtain $\mathcal{C}_0, \ldots, \mathcal{C}_l$. In particular, it follows that $\mathcal{C}_0$ is a Sigma uniform clique cover of G satisfying $\texttt{wgt}(\mathcal{C}_0) \leq |V| + k$. Thus, $(G, |V| + k)$ is a yes-instance.

For the reverse direction, let $\mathcal{C}$ be a Sigma uniform clique cover of G with $\texttt{wgt}(\mathcal{C}) \leq |V| + k$. By iteratively applying Lemma 7.1 for a number of times, call it l, either until a uniform cluster graph is obtained as a direct result of the lemma, or alternatively, stopping after $l = k$ iterations, we can obtain a sequence of graphs $H_0, \ldots, H_l$ and their corresponding sigma uniform clique cover $\mathcal{C}_0, \ldots, \mathcal{C}_l$ respectively.

We shall now verify that also in the latter case where $l = k$, H_l must be a uniform cluster graph. As a consequence of the k applications of Lemma 7.1, we get $\texttt{wgt}(\mathcal{C}_l) \leq |V(H_l)|$.

By considering the fact that for each vertex $v \in V(H_l)$ there exists $C \in \mathcal{C}_l$ with $v \in C$ (since $\mathcal{C}_l$ is a sigma clique cover of H_l), we derive $\text{wgt}(\mathcal{C}_l) \geq |V(H_l)|$.Thus, we have that $\texttt{wgt}(\mathcal{C}_l) = |V(H_l)|$ and it follows that $\mathcal{C}_l$ forms a partition of $V(H_l)$. Using this partition property and the fact that $\mathcal{C}_l$ is a Sigma uniform clique cover of H_l allows us to directly conclude that H_l is a uniform cluster graph. Thus, H_l is a uniform cluster graph in both cases.

□

Theorem 30. UNIFORM CLUSTER-INCLUSIVE VERTEX SPLITTING *parameterized by solution size admits a kernel with at most* $4k$ *vertices.*

Proof. Let $G_0, G_1, G_2, ...G_l$ be a splitting sequence of length l for G. Let $S = \{v_1, v_2, \ldots, v_l\}$ be the collection of vertices that were split to obtain G_l from $G_0 = G$.

Claim 3.2.2. *Let a vertex* v_i *be split into* v_{i1} *and* v_{i2}*. Define* $S_0 = \{v \in G : v \in N(v_{i1}) \cap N(v_{i2}), 1 \leq i \leq l\}$*. Then* $S_0 \subseteq S$*.*

Proof. For the sake of contradiction, let us assume that $v \in S_0$ but $v \notin S$. Since $v \in S_0$, there exists some $v_i \in S$ such that $v \in N(v_{i1}) \cap N(v_{i2})$ for some $1 \leq i \leq l$. By Lemma 26, there exist descendants of v_{i1} and v_{i2}, say v'_{i1} and v'_{i2}, such that $v'_{i1}v \in E(G_l)$ and $v'_{i2}v \in E(G_l)$. Since $v_{i1}v_{i2} \notin E(G_j)$, it follows from Lemma 25 that there is no edge between v'_{i1} and v'_{i2}. Therefore, $G[v'_{i1}, v, v'_{i2}]$ induces a path P_3 in G_l, contradicting the fact that G_l is a cluster graph. Hence $S_0 \subseteq S$. ▷

Next, we observe that if there exists a split sequence of length at most k, then the vertices in $V \setminus S$ have equal degrees in both G and G_l. In other words, for every $x \in V \setminus S$, we have $d_G(x) = d_{G_l}(x)$. We assume that G has at least $2k + 1$ vertices; otherwise, we already have a kernel of size at most $2k$. We know that if the input is a yes-instance, then at least $k + 1$ vertices have the same degree, say d, while at most k vertices have degree different from d. Therefore, we first check whether the input instance (G, k) satisfies this condition. If it does not, we can conclude that the instance is a no-instance. If it does, we can compute the exact value of d in polynomial time. Assuming that the input is a yes-instance, let c be the size of the equal-sized cliques in G_l. One can observe that c must be equal to $d + 1$ in G_l.

Since the graph has no isolated vertices, every vertex must belong to some clique in any Sigma Clique Cover.

Reduction Rule UCIVS 1. *If any of the following conditions hold, we conclude that the instance is a no-instance:*

1. $|\{v \in V(G) \mid d_G(v) > d\}| > k$,
2. *There exists a vertex* $v \in V(G)$ *such that* $d_G(v) < d$,
3. *There exists a vertex* $v \in V(G)$ *such that* $d_G(v) = d$ *and* $G[N[v]]$ *is not a clique.*

Note that the correctness of this reduction rule follows from arguments similar to those used in the proof of Lemma 23.

Lemma 31. *Reduction Rule 1 is correct.*

Proof. We justify the correctness of each condition in Reduction Rule 1 as follows:

1. **Condition 1**: If $|\{v \in V(G) \mid d_G(v) > d\}| > k$, then there are at least $k+1$ vertices whose frequency must be at least two in any sigma $(d+1)$-clique cover. This means for any sigma $(d+1)$-clique cover of G cost must be greater than k. Therefore, the instance must be a no-instance.
2. **Condition 2**: If there exists a vertex $v \in V(G)$ such that $d_G(v) < d$, then clearly $G[N[v]]$ does not contain a clique of size $d+1$. It means v can not be part of a clique in a sigma $(d+1)$-clique cover. Thus, the instance is a no-instance.
3. **Condition 3**: If there exists a vertex $v \in V(G)$ such that $d_G(v) = d$ and $G[N[v]]$ is not a clique. It means $G[N[v]]$ does not contain a clique of size $d+1$ and v can not be part of a clique in a sigma $(d+1)$-clique Cover. Thus, the instance is a no-instance.

Since all three conditions correctly identify invalid instances, the reduction rule is sound. □

Now, we make two cases based on the values of d.

Case 1. Let us assume $d \geq 2k$

In this case, we provide a polynomial time algorithm to solve the problem, that is, due to Lemma 29, we determine whether there exists a sigma $(d+1)$-clique cover of G with cost at most k.

Algorithm 3.2.2: Sigma $(d+1)$-Clique Cover

Require: A graph $G = (V, E)$ and an integer k.
Ensure: A sigma $(d+1)$- clique cover $\mathcal{C}$ of cost at most k, or conclude that no such sigma cover exists.

1: Initialize an empty sigma cover: $\mathcal{C} \leftarrow \emptyset$
2: **while** $G \neq \emptyset$ **do**
3: Find a vertex $v \in V(G)$ such that $d_G(v) = d$
4: **if** no such vertex exists **then**
5: **return** "No instance."
6: **end if**
7: Let $C = G[N[v]]$, the subgraph induced by the closed neighborhood of v
8: **if** C is not a clique **then**
9: **return** "No instance."
10: **end if**
11: Add C to $\mathcal{C}$:

$$\mathcal{C} \leftarrow \mathcal{C} \cup C$$

12: **Check Consistency of $\mathcal{C}$:** Compute $\text{cost}(\mathcal{C}) = \sum_{v \in V} \max(0, \text{freq}_{\mathcal{C}}(v) - 1)$. If $\text{cost}(\mathcal{C}) > k$, return "No instance."
13: **Update G and k:**

$$V(G) \leftarrow V(G) \setminus \{u \in N[v] \mid d_G(u) = d\},$$
$$k \leftarrow k - \text{cost}(\mathcal{C})$$

14: **end while**
15: **return** $\mathcal{C}$

Lemma 32. *The Sigma $(d+1)$-Clique Cover Algorithm (Algorithm 3.2.2) correctly outputs a sigma $(d+1)$-clique cover of cost at most k, or conclude that no such clique cover exists.*

Proof. The correctness of the algorithm is established as follows:

1. **Initial If Conditions**: The first two `if` conditions (lines 4 and 7 of the algorithm) directly follow from **Reduction Rule 1**:

- If no vertex v exists with $d_G(v) = d$, then it is impossible to construct a valid $(d+1)$-sigma clique cover, as per **Condition 1** of the reduction rule.
- If $G[N[v]]$ is not a clique for a vertex v with $d_G(v) = d$, then v cannot belong to any K_{d+1}-clique, as per **Condition 3** of the reduction rule.

2. **Cover Update**: When a vertex v with $d_G(v) = d$ is found, all edges adjacent to v must belong to the same part of the partition $\mathcal{C}$. This is guaranteed by the definition of a $(d+1)$-sigma clique cover, which requires that $C = G[N[v]]$ is a clique. Therefore, adding $C = G[N[v]]$ to $\mathcal{C}$ is a valid and safe update.

3. **Consistency Check**: The **consistency condition** holds due to **Lemma 29**, which guarantees that cost of the partition $\text{cost}(\mathcal{C})$ does not exceed the allowed budget k.

4. **Graph Updates**: Since all edges adjacent to the vertices in the set $\{u \in N[v] \mid d_G(u) = d\}$ have been included in $\mathcal{C}$ and any vertex in the set $\{u \in N[v] \mid d_G(u) = d\}$ can be in only one clique, it is safe to delete these vertices. We delete edges vu for all $u \in N(u)$ because edge vu can be part of only one clique. If w and x are vertices with degree d and $wx \in E(G)$, then we delete wx, because edge wx can be part of exactly one clique. Additionally, the budget k is correctly reduced by the amount $\text{cost}(\mathcal{C})$.

5. **Termination**: The algorithm terminates when G becomes empty. If any condition fails during execution, the algorithm correctly concludes a no-instance.

Since $d \geq 2k$, every vertex of degree greater than d is adjacent to a vertex of degree d otherwise Reduction Rule UCIVS 1 could have been applied. Note that after each iteration, if G is not empty, we will either have a vertex of degree d or all vertices have a degree not equal to d. In the latter case, we return no-instance by Reduction Rule 1 and in the former case, since we have a vertex with degree d, we execute the line 3 of the algorithm. This means that either G becomes empty or any condition fails during execution.

Thus, the algorithm is correct. □

Case 2. Let us assume $d < 2k$.

There is a subtle difference between the two cases. In Case 1, after each iteration of updating G, one of two outcomes occurs: either the graph G becomes empty, or $|V(G)| \geq 2k+1$. In the former case, we can output the partition directly. In the latter case, Reduction Rule 1 guarantees that at least $k+1$ vertices (and all but k) have degree d. This ensures that if the reduced instance is a yes-instance, the equal-sized cliques in the final graph must be of size $d+1$, which is consistent with the original instance.

In Case 2, we must ensure that after each update of G, the graph either becomes empty or contains at least $2k+1$ vertices. Here, Reduction Rule 1 similarly ensures that at least $k+1$ vertices (and all but k) have degree d. Since $d \leq k$, the graph must have at least $4k+1$ vertices to satisfy these conditions. Therefore, we apply the Sigma $(d+1)$-Clique Cover Algorithm (Algorithm 3.2.1) only if $|V(G)| > 4k$.

Reduction Rule UCIVS 2. *If $|V(G)| > 4k$, then run the Sigma $(d+1)$-Clique Cover Algorithm (Algorithm 3.2.2) on the graph G, but terminate the algorithm if the size of the vertex set becomes $|V(G)| \leq 4k$.*

The correctness of the Reduction Rule 2 follows from correctness of Sigma $(d+1)$-Clique Cover Algorithm (Algorithm 3.2.2) as established by Lemma 29. This completes the proof of Theorem 30. □

3.3 Kernelization algorithm for UCEE parameterized by solution size

In this section, we study the following problem: Given a graph G, can we transform G into a uniform cluster graph by editing at most k adjacencies, where editing consists of adding or deleting at most k edges? More formally, let $G = (V, E)$ be a graph. A set $F \subseteq V \times V$ is called a *uniform cluster editing set* for G if $G \triangle F$ is a uniform cluster graph. In this section, we prove the following theorem.

Theorem 33. UCEE *parameterized by solution size admits a kernel of size $\mathcal{O}(k^2)$.*

Note that many preprocessing rules applied in the case of UCEE are also applicable to the cases of UCEA and UCED. To maintain conciseness, we will refer to the following

preprocessing step as the *preparation step*, which begins here. This preprocessing step forms the core idea behind the kernelization algorithms for UCEE, UCED and UCEA.

A graph is a cluster graph if and only if it does not have an induced P_3. It is straightforward to compute a maximal set $\mathcal{P}_3$ of vertex-disjoint induced P_3s in G. If $|\mathcal{P}_3| > k$, then we have a no-instance. Therefore, we assume that $|\mathcal{P}_3| \leq k$, and let S be the vertices of $\mathcal{P}_3$. We have $|S| \leq 3k$. Let $\mathcal{C}$ denote the set of cliques of $G - S$. Let F be a uniform cluster editing set of size at most k for G, and let V_F be the vertices of F. Then we have $|V_F| \leq 2k$.

We observe that if there indeed exists a uniform cluster editing set F of size at most k, then the vertices in $V \setminus V_F$ have equal degree in both G and $G \triangle F$. In other words, for any two distinct vertices $x, y \in V \setminus V_F$, we have $d_G(x) = d_G(y) = d_{G \triangle F}(x) = d_{G \triangle F}(y)$. We assume that G has at least $4k+1$ vertices; otherwise, we would have a kernel of size $4k$. It is known that if the input is a yes-instance, then at least $2k+1$ vertices have the same degree d, and at most $2k$ vertices have degrees not equal to d. Therefore, we first check if the input instance (G, k) satisfies this condition. If the input instance does not satisfy this condition, then conclude that we are dealing with a no-instance. If the input instance satisfies this condition, we can calculate the exact value of d in polynomial time. Assuming that the input is a yes-instance, let c be the size of equal-sized cliques in $G \triangle F$. It follows that c must be equal to $d+1$ in $G \triangle F$. This implies that the minimum degree of G is at least $d-k$, because executing k edge editing operations can increase the degree of a vertex by at most k. Similarly, the maximum degree of G is at most $d+k$, because executing k edge editing operations can decreases the degree of a vertex by at most k. The preparation step ends here.

Next, we provide some Reduction Rules.

Reduction Rule UCEE 1. *If $\delta(G) < d - k$ or $\Delta(G) > d + k$ then conclude that we are dealing with a no-instance.*

Reduction Rule UCEE 2. *If the number of vertices in $V(G)$ with degrees not equal to d exceeds $2k$, then conclude that we are dealing with a no-instance.*

We will consider two cases based on the value of d.

Case 1: Let us assume that $d \geq 6k$.

Lemma 34. *If (G, k) is a yes-instance, then for each $C \in \mathcal{C}$, we have $|C| > 2k$.*

Proof. If there is a clique $C \in \mathcal{C}$ such that $|C| \leq 2k$, then every vertex $v \in C$ has a degree at most $5k - 1$. This is because v has at most $2k - 1$ neighbours within C and at most $3k$ neighbours in S. Given $d \geq 6k$, we have $5k - 1 < d - k$. This implies $\delta(G) < d - k$. Accordingly to Reduction Rule 1, if $\delta(G) < d - k$, then we conclude that we are dealing with a no-instance. Therefore, we must have $|C| > 2k$. □

Reduction Rule UCEE 3. *If $s \in S$ is adjacent to at least $k+1$ vertices of a clique $C \in \mathcal{C}$, then add all missing edges between s and C and decrement the parameter k by $|W|$, where W is the set of all missing edges between s and C. The resulting instance is $(G + W, k - |W|)$. If $|W| \geq k + 1$, then conclude that we are dealing with a no-instance.*

Lemma 35. *Reduction Rule 3 is safe.*

Proof. The vertices of C must belong to the same connected component in the edited graph. Since s has at least $k + 1$ neighbours in C, s must be in the same connected component as the vertices in C. Otherwise, to place s in a separate component, we would need to delete at least $k + 1$ edges between s and C, which contradicts our assumption that $|F| \leq k$. Therefore, we should add all the missing edges between s and C. □

Reduction Rule UCEE 4. *If $s \in S$ is not adjacent to at least $k + 1$ vertices of a clique $C \in \mathcal{C}$, then delete all edges between s and C, and decrement the parameter k by $|W|$, where W is the set of edges between s and C. The resulting instance is $(G \triangle W, k - |W|)$. If $|W| \geq k + 1$, then conclude that we are dealing with a no-instance.*

Lemma 36. *Reduction Rule 4 is safe.*

Proof. All the vertices of C must belong to the same connected component in the edited graph. Note that s cannot be in the same connected component as the vertices in C, as it would require adding at least $k + 1$ edges. Therefore, we must delete all the edges between s and C. □

Observe that exhaustive application of reductions UCEE 3 and UCEE 4 ensures that for each $s \in S$ and each $C \in \mathcal{C}$, either s is adjacent to every vertex of C or s is adjacent to no vertex of C.

Reduction Rule UCEE 5. *If $s \in S$ has neighbours in two distinct cliques $C_1, C_2 \in \mathcal{C}$, then conclude that we are dealing with a no-instance.*

Lemma 37. *Reduction Rule 5 is safe.*

Proof. Due to Reduction rules UCEE 3 and UCEE 4, s must be adjacent to every vertex of C_1 and C_2. This implies that the graph contains at least $2k+1$ edge disjoint induced P_3s. Therefore the input instance is a no-instance. □

Reduction Rule UCEE 6. *If s_1 and s_2 are two non-adjacent vertices of S and have a neighbour in $C \in \mathcal{C}$, then add the edge (s_1, s_2) and decrement the parameter k by 1. The resulting instance is $(G + (s_1, s_2), k-1)$.*

Lemma 38. *Reduction Rule 6 is safe.*

Proof. Due to reductions UCEE 3 and UCEE 4, s_1 and s_2 are adjacent to every vertex of C. Thus, s_1 and s_2 must be in the same connected component as the vertices in C in the edited graph. Therefore, we must add the edge (s_1, s_2) to the solution. □

Reduction Rule UCEE 7. *If s_1 and s_2 are two adjacent vertices of S and have neighbours in $C_1 \in \mathcal{C}$ and $C_2 \in \mathcal{C}$ respectively, then delete the edge (s_1, s_2) and decrement the parameter k by 1. The resulting instance is $(G - (s_1, s_2), k-1)$.*

Lemma 39. *Reduction Rule 7 is safe.*

Proof. The vertices of C_1 and the vertices of C_2 must be in different connected components in the edited graph. Therefore, s_1 and s_2 must also be in different connected components in the edited graph. Therefore, we must delete the edge (s_1, s_2) and add it to the solution. □

Lemma 40. *If (G, k) is a yes-instance and none of the reduction rules UCEE 1 to UCEE 7 are applicable to G, then G is a disjoint union of cliques, each of size $d+1$.*

Proof. When none of the reduction rules UCEE 1 to UCEE 7 are applicable to G, then G is clearly a disjoint union of cliques. Now, we show that these cliques are of size $d+1$. If there

exists a clique of size more than $d+1$, then there would be more than $2k$ vertices whose degree is not equal to d. Similarly, if there exists a clique of size less than $d+1$, then there would be more than $2k$ vertices whose degree is not equal to d, because as shown in Lemma 34, all cliques $C \in \mathcal{C}$ have size at least $2k+1$. This scenario contradicts Reduction Rule UCEE 2. □

In Case 1, we obtain a kernel of size $\mathcal{O}(1)$.

Case 2: Let us assume that $d \leq 6k-1$.

We partition $\mathcal{C} = G - S$ into four parts as follows:

- $\mathcal{C}_< = \{C \in \mathcal{C} : |C| < d+1\}$
- $\mathcal{C}_{0,d+1} = \{C \in \mathcal{C} : |C| = d+1$ and no vertex in C has a neighbour in $S\}$
- $\mathcal{C}_{1,d+1} = \{C \in \mathcal{C} : |C| = d+1$ and some vertex in C has a neighbour in $S\}$
- $\mathcal{C}_> = \{C \in \mathcal{C} : |C| > d+1\}$

Lemma 41. *If (G,k) is a yes-instance and Reduction Rule UCEE 2 is not applicable to G, then $\bigcup_{C \in \mathcal{C}_>} |V(C)| \leq 2k$.*

Proof. Note that every vertex in $\bigcup_{C \in \mathcal{C}_>} V(C)$ has degree more than $d+1$. Due to Reduction Rule 2, the number of vertices with degree more than $d+1$ is bounded by $2k$.

Reduction Rule UCEE 8. *If $|\mathcal{C}_{0,d+1}| > 2k$, then retain only $2k+1$ cliques from $\mathcal{C}_{0,d+1}$ and discard the rest.*

Lemma 42. *Reduction Rule 8 is safe.*

Proof. An edge editing set of size at most k can change degrees of at most $2k$ vertices in $\mathcal{C}_{0,d+1}$. In other words, an edge editing set of size at most k can affect at most $2k$ cliques in $\mathcal{C}_{0,d+1}$. There will still exist at least one clique of size $d+1$ in the final graph. Therefore, we can retain only $2k+1$ cliques. □

Lemma 43. $|\mathcal{C}_{1,d+1}| \leq 2k$.

Proof. Each clique $C \in \mathcal{C}_{1,d+1}$, by definition, contains at least one vertex that has a neighbour in S. This implies that every clique in $\mathcal{C}_{1,d+1}$ contains at least one vertex whose degree is more than d. Due to Reduction Rule UCEE 2, there are at most $2k$ such vertices. Therefore, $|\mathcal{C}_{1,d+1}| \leq 2k$. □

Therefore, we get $\left|\bigcup_{C \in \mathcal{C}_{d+1}} V(C)\right| \leq (4k+1)(6k-1) = 24k^2 + 2k - 1$, where $\mathcal{C}_{d+1} = \mathcal{C}_{0,d+1} \cup \mathcal{C}_{1,d+1}$.

Lemma 44. *If (G, k) is a yes-instance and Reduction Rule UCEE 1 is not applicable to G, then we have $\left|\bigcup_{C \in \mathcal{C}_<} V(C)\right| < 21k^2 + 5k$.*

Proof. Since Reduction Rule UCEE 1 is not applicable to G, every vertex in G has a degree at most $d + k \leq 7k - 1$. For the sake of contradiction, assume that $\bigcup_{C \in \mathcal{C}_<} |V(C)| \geq 21k^2 + 5k$. Each clique $C \in \mathcal{C}_<$ has size less than $d + 1$, hence the vertices in C have degree less than d. As (G, k) is a yes-instance, G has a uniform cluster edge editing set of size at most k that can increase the degree of at most $2k$ vertices. Therefore, at least $21k^2 + 5k - 2k$ vertices in $\bigcup_{C \in \mathcal{C}_<} V(C)$ have neighbours in S. Since $|S| \leq 3k$ and S has at least $21k^2 + 3k$ neighbours in $G - S$, by the Pigeonhole principle, there is a vertex $s \in S$ with at least $7k + 1$ neighbours in $G - S$, which contradicts the fact that $d_G(v) \leq 7k - 1$ for all $v \in V$. Therefore, we conclude that $\bigcup_{C \in \mathcal{C}_<} |V(C)| < 21k^2 + 5k$. □

Applying the above reduction rules and results, we find that:

$$\begin{aligned}
|V(G)| &= |S| + |V \setminus S| \\
&= |S| + \left|\bigcup_{C \in \mathcal{C}_<} V(C)\right| + \left|\bigcup_{C \in \mathcal{C}_{d+1}} V(C)\right| + \left|\bigcup_{C \in \mathcal{C}_>} V(C)\right| \\
&\leq 3k + (21k^2 + 5k) + (24k^2 + 2k - 1) + 2k \\
&= 45k^2 + 12k - 1.
\end{aligned}$$

3.4 Parameterized Complexity of UCED and UCEA

In this section, we see a number of parameterized complexity results regarding the complexity of both the UCED and UCEA.

3.4.1 A Kernelization Algorithm for UCED

In this section, we provide kernelization algorithms for UCED.

Theorem 45. *The* UCED *problem parameterized by solution size admits a kernel with at most* $6k$ *vertices.*

Proof of Theorem 45. We start with the same preparation step as designed above. However, in this preparation step, we apply edge deletion operations instead of edge editing operations.

Reduction Rule UCED 1. *If* $\delta(G) < d$ *or* $\Delta(G) > d+k$*, then conclude that we are dealing with a no-instance.*

Reduction Rule UCED 2. *If the number of vertices in* $V(G)$ *with degrees not equal to* d *exceeds* $2k$*, then conclude that we are dealing with a no-instance.*

Reduction Rule UCED 3. *If there exists a vertex* u *of degree* d *in* G *such that the subgraph induced by* $N[u]$ *is not a clique, then conclude that we are dealing with a no-instance.*

Lemma 46. *Reduction Rule 3 is safe.*

Proof. Note that if the input is a yes-instance, then every vertex u of degree d must be part of a clique of size $d+1$ after deleting the edges of the solution. Since the degree of u is d, this can only happen if the closed neighborhood of u (denoted $N[u]$) forms a clique. Therefore, if the subgraph induced by $N[u]$ is not a clique, the instance is a no-instance. □

Reduction Rule UCED 4. *Let* $u \in V(G)$ *be a vertex with degree* d*. If* u *has a neighbour* v *with a degree strictly greater than* d*, then delete all the edges* $(v, w) \in E(G)$ *such that* w *is neither* u *nor a neighbour of* u*. The set of edges deleted is denoted as* W*. The resulting instance is* $G = (V(G), k - |W|)$*. Also if* $|W| \geq k+1$ *then conclude that we are dealing with a no-instance.*

Lemma 47. *Reduction Rule 4 is safe.*

Proof. As we have seen from the proof of Lemma 46, $N[v]$ must form a clique component in the final graph. Therefore, we must delete the edges $(v, w) \in E(G)$ such that $v \in N[u]$ but $w \in V(G) \setminus N[u]$. It is also worth noting that the condition $|V(G)| \geq 4k + 1$ ensures that the resulting graph, obtained by deleting the edges in the solution, must be a cluster graph, where every clique is of size exactly $d + 1$. □

Reduction Rule UCED 5. *If there are multiple cliques of size $d + 1$, then we retain only x cliques of size $d+1$ where $x(d+1) \geq 2k+1$ and remove the remaining cliques of size $d+1$.*

Lemma 48. *Reduction Rule 5 is safe.*

Proof. The condition $x(d + 1) \geq 2k + 1$ in Reduction Rule UCED 5 ensures that at least $2k + 1$ vertices have degree exactly d, which in turn ensures that the size of cliques, after deleting at most k edges, remains exactly $d + 1$. Therefore deleting the extra cliques does not change the size of a solution. □

We consider two cases based on the size of d.

Case 1: $d \geq 2k$

Lemma 49. *After applying Reduction Rules 1-5 exhaustively, we obtain a uniform cluster graph.*

Proof. First we will show that every vertex in the reduced graph has degree d. For the sake of contradiction assume there exists a vertex v with a degree of at least $d+1$. If $N(v)$ contains a vertex with degree d, then Reduction Rule UCED 4 could have been applied. Therefore, we assume that all vertices in $N(v)$ have a degree of at least $d + 1$. As a result, the number of vertices with degrees not equal to d is at least $d + 2 \geq 2k + 2$ as $d \geq 2k$. However, in this scenario, Reduction Rule UCED 2 could have been applied because the number of vertices with degrees not equal to d exceeds $2k$. Consequently, in a reduced graph, every vertex has a degree exactly d. After exhaustively applying Reduction Rules UCED 3, 4 and 5, we obtain a clique of size $d + 1$. This is a uniform cluster graph. □

Therefore in the case 1, we obtain a kernel of size $\mathcal{O}(1)$.

Case 2: $d < 2k$

After exhaustively applying Reduction Rules UCED 3 and UCED 4, the components containing a vertex of degree d must form a $d+1$-sized clique. After applying UCED 5, the cliques of size $d+1$ collectively contain at most $4k$ vertices. According to Reduction Rule UCED 2, we have at most $2k$ vertices that do not have degree d. Therefore, we have $|V(G)| \leq 6k$.

This completes the proof of Theorem 45.

3.4.2 A Kernelization Algorithm for UCEA

In this section, we provide kernelization algorithms for UCEA.

Theorem 50. *The* UCEA *problem admits a kernel with at most* $5k$ *vertices.*

Proof of Theorem 50. We start with the same preparation step. However, in this preparation step, we apply edge addition operations instead of edge editing operations.

Reduction Rule UCEA 1. *If* $\delta(G) < d-k$ *or* $\Delta(G) > d$*, then conclude that we are dealing with a no-instance.*

Reduction Rule UCEA 2. *If the number of vertices in* $V(G)$ *with degrees not equal to* d *exceeds* $2k$*, then conclude that we are dealing with a no-instance.*

Reduction Rule UCEA 3. *If there is a path between two nonadjacent vertices* u *and* v*, then make* u *adjacent to* v *and decrease* k *by 1. The new instance becomes* $(G+(u,v), k-1)$.

Lemma 51. *Reduction Rule 3 is safe.*

Proof. Since there is a path between u and v, they must belong to the same clique in the edited graph. Therefore, we must connect u and v by an edge. □

Observe that exhaustive application of Reduction Rule UCEA 3 ensures that the new graph is a collection of disjoint cliques.

Reduction Rule UCEA 4. *If there are multiple cliques of size $d+1$, then we retain only x cliques of size $d+1$ where $x(d+1) \geq 2k+1$ and remove the remaining cliques of size $d+1$.*

Lemma 52. *Reduction Rule 4 is safe.*

Proof. Note that Reduction Rule UCEA 4 ensures that at least $2k+1$ vertices have a degree exactly d, which in turn ensures that the size of cliques remains exactly $d+1$ after adding at most k edges. Therefore, deleting the extra cliques does not change the size of the solution. □

Case 1: $d \geq k+1$

Reduction Rule UCEA 5. *If there is a clique of size less than $d+1$ in G, then conclude that we are dealing with a no-instance.*

Proof. Let us assume that we have a clique C of size $x < d+1$. To obtain a clique of size $d+1$ from C, we would need to add at least $x(d+1-x)$ edges. However, this exceeds the allowable number k of edge addition operations, as $x(d+1-x) > k$. □

Note that due to Reduction Rule UCEA 1, there cannot be a clique of size more than $d+1$. Hence applying Reduction Rules UCEA 1 to UCEA 5, all cliques are of size $d+1$.

In Case 1, we obtain a kernel of size $\mathcal{O}(1)$.

Case 2: $d \leq k$

After applying Reduction Rule UCEA 4, the cliques of size $d+1$ collectively contain at most $3k$ vertices. Accordingly to Reduction Rule 2, there are at most $2k$ vertices with degrees not equal to d. Therefore, we have $|V(G)| \leq 5k$.

This completes the proof of Theorem 50.

3.4.3 An FPT Algorithm for UCED

Böcker and Damaschke [20] provided an FPT algorithm for the Cluster Edge Deletion problem. This algorithm is based on branching technique. The core idea is to apply a branching rule to eliminate all induced P_3 in the graph. Once no further applications of the branching rules are possible, the remaining instance can be solved in polynomial time. Since the Uniform Cluster Edge Deletion problem also requires elimination of all induced P_3's, the same branching rules can be applied in our case as well. Once the branching rules are no longer applicable, we again solve the remaining instance in polynomial time. For completeness, we explain the terminology and branching rules introduced by Böcker and Damaschke. The score of an edge e is defined as the number of induced P_3 in which e appears. A graph is said to be score-s if every edge in the graph has a score of at most s. The score of a graph is the maximum score among all its edges. Branching on an edge e means either (i) deleting the edge e, or (ii) deleting all other edges that, together with e, form an induced P_3.

The operation of *doubling* a vertex x in a graph refers to inserting a new vertex that is adjacent exactly to x and all its neighbors. The result of doubling multiple vertices is independent of the order in which the vertices are doubled. Below are definitions of several special graphs on six vertices, where only the explicitly mentioned edges exist.

- 3-asterisk: a K_3 where each vertex is adjacent to one additional (distinct) vertex.
- 3-sun: a K_3 where each pair of vertices shares an additional adjacent vertex.
- fat P_4 : A path on four vertices (P_4) where both inner vertices have been doubled.
- fat P_5: A path on five vertices (P_5) where the central vertex has been doubled.

The *disjoint union* $G + H$ of two graphs G and H is the graph consisting of disjoint copies of G and H. The notation pG denotes the disjoint union of p copies of G. The *join* of two graphs G and H, denoted by $G * H$, is obtained by taking $G + H$ and adding all possible edges between vertices of G and vertices of H. Now, we state the theorem provided by Damaschke [41].

Theorem 53. *[41] The following graphs (with arbitrarily large positive n, q, p) and their connected induced subgraphs comprise the complete list of connected score-2 graphs: 3-asterisk,*

3-sun, fat P_4, fat P_5; C_n for $n \geq 4$, $K_q * C_5$, $K_q * K_3^c$, $K_q * (K_2 + K_2)$, and $(qK_1 + pK_2)^c$ for $p \geq 2$.

Lemma 54. *If an induced C_4 exists, we can apply a 1.415 rule for Uniform Cluster Edge Deletion.*

Proof. If we delete at most one edge from each pair of opposite edges in an induced C_4 then, obviously, some induced P_3 remains. Thus we must delete some pair of opposite edges, which yields the branching vector $(2, 2)$. □

In [20] Böcker and Damaschke gave an FPT algorithm for cluster editing. The idea of the FPT algorithm was to first branch on the edge of a score larger than 2. Then in the graph of score-2, use the classification given by the Theorem 53 and deal with each connected component separately. Our algorithm will also start with branching on the edge of a score larger than 2. But for our problem, we will have to be careful about the size of each component. To deal with our problem, we will first guess the size of equal-sized cliques and deal with each component according to the guessed size.

Theorem 55. Uniform Cluster Edge Deletion *is solvable in $\mathcal{O}(1.47^k)$ time.*

Proof. We know that deleting the vertices of any solution results in a disjoint union of equal-sized cliques (say, of size c). We start with guessing the size c. Note that the possible guesses for c range from 1 to n. If c is a valid guess, then the number of vertices in G must be a multiple of c.

As long as possible, take an edge e of score larger than 2, and delete e or all edges forming a P_3 with e. The branching number is 1.47. Once the graph is score-2, every component is one of the graphs from the Theorem 53. Now we deal with each component to get a union of c-sized cliques.

- Note that the problem can be solved in polynomial time on 3-asterisk, 3-sun, fat P_4, fat $P_5, bC_n (n \geq 4)$.
- In $(qK_1 + pK_2)^c$, we note that if $p \geq 2$, then we have an induced C_4. In this case, we can apply the 1.415 branching rule in Lemma 54. Therefore, we only need to deal

with the case when $p = 1$. If $p = 1$, then then the component is actually an induced subgraphs of $K_q * K_3^c$.

- The other graphs in the theorem consist of one clique K of size q, joined with at most five other vertices. Suppose our connected component is $K_q * H$ where $|V(H)| \leq 5$. We note that $q + |V(H)|$ is a multiple of c. Otherwise we can return a no instance.

  Suppose F be our desired solution. Then $G - F$ must be disjoint union of c-sized cliques, say $C_1, C_2, \ldots, C_{\frac{q+|V(H)|}{c}}$.

  We guess a partition $\mathcal{P} = \{P_1, P_2, \ldots, P_l\}$ of vertices of H. We say a partition $\mathcal{P} = \{P_1, P_2, \ldots, P_l\}$ corresponds to a solution F if $P_i = V(C_i) \cap V(H)$ and $G[P_i]$ is a clique. Otherwise $\mathcal{P}$ is a wrong guess. For each part P_i, there must exist $U_i \subseteq V(K_q)$ such that $G[P_i \cup U_i] = C_i$. We can easily find U_i in polynomial time. This is because the vertices in K_q are true twins.

  Note that $l \leq 5$. We take any arbitrary partition of $V(K_q)$, say $\mathcal{U} = \{U_1, U_2, ..., U_r\}$ such that $|U_i| = c - |P_i|$ for $1 \leq i \leq l$ and $|U_i| = c$ for $l + 1 \leq i \leq \frac{q+|V(H)|}{c}$.

Define

$$P_i' = \begin{cases} P_i \cup U_i \text{ for } 1 \leq i \leq l \\ U_i \text{ for } l+1 \leq i \leq \frac{q+|V(H)|}{c} \end{cases}$$

Now we have $\mathcal{P}' = \left\{ P_1', P_2', \ldots, P_{\frac{q+|V(H)|}{c}}' \right\}$ a partition of vertices of the connected component. If for some i, $G[P_i']$ is not a clique, we discard the partition $\mathcal{P}$. If every $G[P_i']$ is a clique, we say that $\mathcal{P}$ is a *valid* partition. For a valid partition, we find the number of edge deletions required to get the partition $\mathcal{P}'$ such that there is no edge between $G[P_i']$ and $G[P_i']$ for $i \neq j$. Suppose $k_{\mathcal{P}}$ denotes the number of edges deleted to get the partition $\mathcal{P}'$ for partition $\mathcal{P}$. Note that once we fixed c, the number $k_{\mathcal{P}}$ is also fixed. For a guess c, We do the above process for all partitions of $V(H)$. Therefore, we solve the problem in polynomial time. □

3.4.4 A Subexponential algorithm for UCED on Dense Graphs

In this section, we will provide a subexponential algorithm for UCED on everywhere α-dense graphs. Let us begin by recalling the definition of everywhere α-dense graphs.

Definition 29 ([8, 84]). *A graph on n vertices is α-dense if it has $\alpha n^2/2$ edges. It is everywhere-α-dense if the minimum degree is αn. We abbreviate $\Omega(1)$-dense as* dense *and everywhere-$\Omega(1)$-dense as* everywhere-dense.

We will use the subexponential algorithm of Lochet et al. for the d-Way Cut problem on everywhere α-dense graphs as a subroutine in our algorithm. Specifically, they proved the following:

Theorem 56. *[90] The d-Way Cut problem, parameterized by the size of the solution, admits an algorithm with running time $\left(\frac{1}{\alpha}\right)^{\mathcal{O}((1/\alpha)^3)} \cdot n^2 \cdot 2^{\mathcal{O}\left(\sqrt{\frac{k}{\alpha}\log\left(\frac{k}{\alpha}\right)}\right)}$ on everywhere-α-dense graphs.*

We now state the main result of this section.

Theorem 57. *The UCED problem, parameterized by the size of the solution, admits an algorithm with running time $\left(\frac{1}{\alpha}\right)^{\mathcal{O}((1/\alpha)^3)} \cdot n^2 \cdot 2^{\mathcal{O}\left(\sqrt{\frac{k}{\alpha}\log\left(\frac{k}{\alpha}\right)}\right)}$ on everywhere α-dense graphs.*

Proof. We will assume that $(\alpha n)^2 \geq 49k$. If not then $n \leq \frac{7\sqrt{k}}{\alpha}$ allowing us to try all possible vertex partitions and solve the problem in the required time.

Let the input instance be a yes-instance of UCED. Suppose that the graph G can be transformed into a uniform cluster graph by deleting a set $F \subseteq E(G)$ of size at most k, resulting in cliques of size $h+1$. We begin by guessing h, which has at most n possible values. Assume h has been correctly guessed. We distinguish between two cases:

Case 1: $h + 1 < \alpha n - \sqrt{k}$.
Since every vertex $v \in V(G)$ has degree $d(v) \geq \alpha n$, it must be adjacent to at least $\sqrt{k}$ edges in F to reduce its degree to h. Thus, we must have $\frac{n\sqrt{k}}{2} \leq k$, which implies $n \leq 2\sqrt{k}$. In this case, we can solve the problem within the required time by exhaustively checking all vertex partitions.

Case 2: $h+1 \geq \alpha n - \sqrt{k}$.

Since $\alpha n \geq 7\sqrt{k}$, we conclude that $h+1 \geq 6\sqrt{k}$. Let us denote by L the set of vertices which are adjacent to at least $\sqrt{k}$ edges in F. We have $L = \{v \in V(G) \mid d(v) \geq h + \sqrt{k}\}$. Therefore $|L| \leq 2\sqrt{k}$ because $\frac{|L| \cdot \sqrt{k}}{2} \leq k$.

We have $G - F$ as a uniform cluster graph containing $\frac{n}{h+1}$ cliques of size exactly $h+1$. Denote by $C_1, C_2, \ldots, C_{\frac{n}{h+1}}$ the cliques in $G - F$ with $|C_i| = h+1$ for all $1 \leq i \leq \frac{n}{h+1}$. Define $C'_i = C_i \setminus L$. Now, construct an instance $I' = (G - L, d, k)$ of d-Way Cut, where $d = \frac{n}{h+1}$. Since I is a yes-instance of UCED, I' is a yes-instance of d-Way Cut. To apply the algorithm in Theorem 56, we need to make sure that the graph $G - L$ is everywhere dense. We can assume that $\alpha n - 2\sqrt{k} > \frac{\alpha n}{2}$. Otherwise, we get $n \leq \frac{4\sqrt{k}}{\alpha}$. In this case, we can try all the partitions and solve the problem in required time. Also, each vertex in G has degree at least αn. Therefore, the degree of every vertex in $G - L$ is at least $\alpha n - 2\sqrt{k} \geq \frac{\alpha n}{2}$. Therefore, we can assume that $G - L$ is everywhere $\frac{\alpha}{2}$-dense. Using the algorithm in Theorem 56, we obtain a set of edges $E' \subseteq E(G - L)$ of minimum size $k' \leq k$ such that $G - L - E'$ has exactly d connected components.

Lemma 58. *The d connected components obtained by deleting E' from $G - L$ are precisely $C'_1, C'_2, \ldots, C'_d$.*

Proof. Let $A_1, A_2, \ldots, A_d$ denote the connected components obtained by removing E' from $G - L$. Each C'_i satisfies $|C'_i| \geq 4\sqrt{k}$ since $|C_i| = h+1 \geq 6\sqrt{k}$ and $|L| \leq 2\sqrt{k}$. As we are deleting at most $k' \leq k$ edges in $G - L$, all but at most $\sqrt{k}$ vertices in C'_i are in the some connected component A_j for some $1 \leq j \leq d$. Suppose C'_i has all but at most $\sqrt{k}$ of its vertices in a component A_j.

Claim 3.4.1. $C'_i \subseteq A_j$

Proof. Assume for contradiction that there exists $x \in C'_i$ with $x \in A_l$ for some $l \neq j$. Construct a new partition $B_1, B_2, \ldots, B_d$ by moving x from A_l to A_j. Let us denote by E_B the set of edges needed to delete to get the connected components as $B_1, \ldots, B_d$. Note that $E_B = (E' \setminus \{(x, w) \mid w \in A_j\}) \cup \{(x, u) \mid u \in A_l\}$. Therefore, $|E_B| = |E'| - d_{A_j}(x) + d_{A_l}(x)$. Note that as $x \notin L$, we know that x has less than $\sqrt{k}$ neighbors outside C_i. Therefore, we have $d_{A_l}(x) \leq \sqrt{k}$. We also know that x has at least $3\sqrt{k}$ neighbors in A_i. Therefore, We have $d_{A_j}(x) \geq 3\sqrt{k}$. This implies that $d_{A_j}(x) > d_{A_l}(x)$. Hence, we get $|E_B| < |E'|$. This is a contradiction to minimality of E'. Therefore, we know that for every $i \in [d]$, there is an

unique $j \in [d]$ such that $C'_i \subseteq A_j$. □

Since all the connected components A_j's are non-empty, for every $i \in [d]$, there is an unique $j \in [d]$ such that $C'_i = A_j$. This finishes the proof of the lemma. □

Due to the previous lemma, we exactly know the partition of all the vertices in graph G except the vertices in L. But as we know that $|L| \leq 2\sqrt{k}$, we can guess for each vertex in L its position in A_i's. As $d = \frac{n}{h+1} \leq \frac{n}{\alpha n - \sqrt{k}} \leq \frac{n}{\left(\frac{\alpha}{2}\right)n} \leq \frac{2}{\alpha}$. This requires at most $\left(\frac{2}{\alpha}\right)^{\sqrt{k}}$ guesses. By trying all the possibilities, we can solve the problem in required time.

This completes the proof of Theorem 57. □

3.5 Structural Parameterization

Apart from considering the solution size as a parameter, we also investigate several structural parameters of the input graph, including *feedback vertex set*, *vertex cover*, and *treewidth*. These parameters capture different aspects of the graph's structure and have been widely used in the parameterized complexity literature to obtain more fine-grained insights into hard problems. Our analysis along these lines provides a natural starting point for a broader exploration of the problem from the perspective of structural parameterizations. Importantly, this line of inquiry is far from exhaustive.

3.5.1 Structural Parameterization of UCVD

Theorem 59. UCVD *parameterized by vertex cover admits a kernel of size* $3\mathsf{vc}(G)$.

Let C denotes a minimum vertex cover of G which is of size $\mathsf{vc}(G)$. We know that the size of an optimal solution is upper bounded by $\mathsf{vc}(G)$ as deleting all vertices in C from G creates cliques of size exactly one. Let $I = G - C$ be an independent set.

Reduction Rule UCVD 10. *If I contains at least $2\mathsf{vc}(G)+1$ vertices then return $\mathsf{vc}(G)$.*

Lemma 60. *Reduction Rule UCVD 10 is safe.*

Proof. Let us assume that I contains at least $2\mathsf{vc}(G)+1$ vertices. Let S be any solution. Clearly, every vertex of $I \setminus S$ must be in distinct clique in $G-S$. Therefore, the size of cliques in $G-S$ must be bounded by one as optimal solution size is at most $\mathsf{vc}(G)$. Therefore, we must delete minimum number of vertices to get a cliques of size exactly one. Clearly, we return the solution of size $\mathsf{vc}(G)$. □

Due to Reduction Rule UCVD 10, we can assume that I contains at most $2\mathsf{vc}(G)$ many vertices. Therefore, we get a kernel of size $3\mathsf{vc}(G)$. This completes the proof of Theorem 59.

Given that UCVD has a kernel of size $3\mathsf{vc}(G)$, one can run a greedy algorithm to solve the problem, which results in the following corollary.

Corollary 3.5.1. UCVD *parameterized by vertex cover can be solved in time $2^{\mathcal{O}(\mathsf{vc}(G))} \cdot n^{\mathcal{O}(1)}$.*

3.5.2 Structural Parameterization of UCED

Theorem 61. UCED *parameterized by vertex cover admits a kernel of size $2\mathsf{vc}(G)$.*

Let C denotes a vertex cover of G which is of size at most $\mathsf{vc}(G)$. Let $I = G - C$ be an independent set.

Reduction Rule UCED 6. *If I contains at least $\mathsf{vc}(G)+1$ vertices then return $|E(G)|$.*

Lemma 62. *Reduction Rule 6 is safe.*

Proof. Let us assume that I contains at least $\mathsf{vc}(G)+1$ vertices. Let S be any solution. Clearly, every vertex of I must be in distinct clique in $G-S$. Therefore, the size of cliques in $G-S$ must be bounded by 1. Therefore, we must delete all the edges to get a cliques of size exactly one. □

Due to Reduction Rule 6, we can assume that I contains at most $\mathsf{vc}(G)$ many vertices. Therefore, we get a kernel of size $2\mathsf{vc}(G)$.

Theorem 63. UCED *parameterized by feedback vertex set admits a kernel of size* $3\mathsf{fvs}(G)+2$.

Let Z denotes a feedback vertex set of G which is of size at most $\mathsf{fvs}(G)$. We denote the forest obtained by deleting the vertices in Z as F.

Reduction Rule UCED 7. *Let us assume that F contains an independent set I of size at least $\mathsf{fvs}(G)+1$. If G contains a perfect matching then return $|E(G)|-\frac{|V(G)|}{2}$ or else return $|E(G)|$.*

Lemma 64. *Reduction Rule 7 is safe.*

Proof. Let S be any solution. Clearly, every vertex of I must be in a distinct clique in $G-S$. As there are more than $\mathsf{fvs}(G)+1$ vertices in I, there exists at least one clique in $G-S$ which contains vertices only from F. Clearly, size of such a clique must be bounded by 2. This implies that all the cliques in $G-S$ must have size exactly one or two. Note that the latter case is only possible when the graph contains perfect matching. We can determine whether the graph contains perfect matching or not in polynomial time. If such a perfect matching exists then we get a solution of size $|E(G)|-\frac{|V(G)|}{2}$. Otherwise, the size of cliques must be one. In this case, we must delete all the edges. Therefore, we can return a solution of size $|E(G)|$. □

As we know that every tree on n vertices contains an independent set of size at least $\lfloor\frac{n}{2}\rfloor$. After applying the above reduction rule, we know that F contains at most $2\mathsf{fvs}(G)+2$ vertices. Therefore, we get a kernel of size $3\mathsf{fvs}(G)+2$.

3.5.3 Structural Parameterization of UCEA

Theorem 65. UCEA *can be solved in time* $\mathcal{O}(\mathsf{vc}^{\mathsf{vc}})$

Proof. We find a vertex cover C of size at most $\mathsf{vc}(G)$. Adding the edges of the solution results in a disjoint union of equal-sized cliques. Suppose vertices of C are parts of q different cliques after adding the edges of the solution. We guess a partition $\mathcal{P}=\{P_1,P_2,\ldots,P_q\}$ of C. If $G[P_i\cup P_j]$ is a connected graph, then $\mathcal{P}$ is a wrong guess. So, we assume that every vertex of $G-C$ is adjacent to atmost one $P_i\in\mathcal{P}$. There are at most $\mathsf{vc}(G)^{\mathsf{vc}(G)}$ candidates for $\mathcal{P}$ and we try out all guesses. □

3.5.4 Structural Parameterization of UCEE

Theorem 66. UCEE *parameterized by vertex cover admits an FPT algorithm with running time* $\mathsf{vc}(G)^{\mathcal{O}(\mathsf{vc}(G)^2)} \cdot n^{\mathcal{O}(1)}$.

Proof. First, we note that any graph whose vertex cover is $\mathsf{vc}(G)$ contains at most $\mathsf{vc}(G).n$ edges. Let F be any solution of minimum size. We make a simple yet important observation that $|F| \leq \mathsf{vc}(G).n$. This is because deleting all the edges in the graph G contains equal cliques of size exactly one.

Claim 3.5.1. *The equal-sized cliques in* $G\Delta F$ *are of size at most* $4.\mathsf{vc}(G)$.

Proof. For the sake of contradiction assume that the cliques in $G\Delta F$ are of size $4.\mathsf{vc}(G)+1$. In this case, let us denote the total number of edges in $G\Delta F$ by l.

$$\begin{aligned} l &\geq \binom{4.\mathsf{vc}(G)+1}{2} \times \frac{n}{4.\mathsf{vc}(G)} \\ &\geq \frac{4.\mathsf{vc}(G)(4.\mathsf{vc}(G)+1)}{2} \times \frac{n}{4.\mathsf{vc}(G)} \\ &\geq \frac{(4.\mathsf{vc}(G)+1)}{2} \times n. \end{aligned}$$

Clearly, this suggests that

$$\begin{aligned} |F| &\geq l - \mathsf{vc}(G).n \\ &\geq \frac{(4.\mathsf{vc}(G)+1)}{2} \times n - \mathsf{vc}(G).n \\ &\geq \frac{(2.\mathsf{vc}(G)+1)}{2} \times n \\ &> \mathsf{vc}(G) \times n. \end{aligned}$$

This is a contradiction. This finishes the proof of claim.

□

Let us denote by H the graph induced by vertices in cliques in $G\Delta F$ that contain the vertices from the vertex cover. The size of cliques in $G\Delta F$ is denoted by q. We will start by making a guess for the value of q. If q does not divide n, we discard such a guess. According

to the previous claim, $|V(H)| \leq 4\mathsf{vc}(G)^2$. Therefore, we can make at most $2^{\mathcal{O}(\mathsf{vc}(G)^2)}$ guesses for the graph H. For each guess, we can determine in polynomial time whether H is indeed an induced subgraph of G or not, with the vertices in the vertex cover of G mapped to themselves. If not, we discard such a guess. If yes, we calculate the minimum number of edges needed to be edited in order to turn H into equal-sized cliques of size q by guessing the partition of H in polynomial time. This process requires at most $\mathsf{vc}(G)^{\mathcal{O}(\mathsf{vc}(G)^2)}$ many guesses. Once that is accomplished, we can select an arbitrary partition of vertices in $I \setminus V(H)$, where each part contains exactly q vertices. Again, we can calculate the minimum number of edges needed to be edited to convert it into a graph of q-sized cliques in polynomial time. Therefore, for each guess, we can solve the problem in polynomial time. In the end, we output the minimum number of edges needed to be edited in order to convert it into a uniform cluster graph. □

Theorem 67. UCEE *parameterized by treewidth admits an XP algorithm with running time* $n^{\mathcal{O}(2^{\mathsf{tw}(G)^2})} \cdot n^{\mathcal{O}(1)}$.

Proof. We know that any graph with treewidth $\mathsf{tw}(G)$ contains at most $\mathsf{tw}(G).n$ edges. As proved in Theorem 66, we can see that any optimal solution F has size at most $\mathsf{tw}(G).n$. Also the size of cliques in $G\Delta F$ is upper bounded by $4\mathsf{tw}(G)$. We begin the algorithm by guessing the size of cliques. Let us denote it by q. Note that q must divide n otherwise it is an incorrect guess. Let us denote the partition of vertices in $G\Delta F$ by $P = \{P_1, P_2, \ldots, P_{\frac{n}{q}}\}$. Let us denote by $F_{del} \subseteq F$ the edges that are deleted from G in the solution. Note that, every connected component in $G - F_{del}$ must be of size at most q. We know that the the maximum number of graphs on q vertices is at most 2^{q^2}. Let us denote the possible graphs using $X_1, X_2, \ldots, X_{2^{q^2}}$. Therefore, we can guess the structure of the graph $G[P_i]$ in $G - F_{del}$ for all i as there are at most $n^{\mathcal{O}(q^2)}$ choices . For every choice, we can easily calculate the number of edges need to be edited in order to get uniform cluster graph. We just need to check whether the graph $H = \bigcup_{i=1}^{\frac{n}{q}} G[P_i]$ is a subgraph of G or not. It is easy to see that one can easily construct a standard dynamic programming algorithm running in time $f(\mathsf{tw}(G)).n^{\mathcal{O}(1)}$ to check whether H is a subgraph of G or not where f is some computable function. For all valid choices, we return the minimum number of edges required to edit in order to get a uniform cluster graph. □

3.6 Conclusions and Open Problems

In this chapter, we studied the problem of modifying a given graph so that the resulting graph becomes a collection of equal-sized cliques. We provided polynomial kernels for various types of modification problems along with FPT algorithms for few variants. While we have made significant progress in understanding the parameterized complexity of the problems studied in this chapter, several intriguing questions remain open for future research:

- Does there exist a linear or quadratic vertex kernel for UCVD?
- Can we design a $(2-\epsilon)^k \cdot n^{O(1)}$ algorithm for UCVD?
- Is there a $2^{O(k)} \cdot n^{O(1)}$ algorithm for UCEVS and UCIVS?
- Does a sub-exponential algorithm for UCEVS and UCIVS exist on everywhere dense graphs?
- Does there exist a linear vertex kernel for UCEE?
- Construct an efficient FPT algorithm for UCEE and UCEA?
- Does a sub-exponential algorithm for UCEE exists on everywhere dense graphs?

These questions represent exciting challenges in the parameterized complexity landscape and provide avenues for further exploration.

While our work provides initial insights into the complexity of the problem under various structural parameters such as feedback vertex set, vertex cover, and treewidth, it only scratches the surface of what is possible within this framework. Given the richness of structural graph parameters in the literature, future investigations could consider other measures such as *clique-width*, *modular-width*, or *neighborhood diversity*, potentially leading to new algorithmic strategies or complexity classifications. Thus, our work serves as a foundation for further exploration and highlights the value of structural parameters in advancing the understanding of this problem.

Chapter 4

Maximum Minimal Degree-d-Modulator

In this chapter, we investigate the MaxMin Degree-d-Modulator, a natural generalization of the well-studied MaxMin Vertex Cover. Given a graph $G = (V, E)$ and an integer $d \geq 0$, a *degree-d-modulator* is a subset $S \subseteq V$ such that the removal of S from G results in a graph where every remaining vertex has degree at most d. A degree-d-modulator is said to be *minimal* if no proper subset of it is a degree-d-modulator. In other words, $G - (S \setminus \{v\})$ contains at least one vertex of degree $d + 1$ or higher. It is easy to observe that if S is a minimal degree-d-modulator, then for every $v \in S$, one of the following holds:

- v has at least $d + 1$ neighbours in $G - S$, or
- v is adjacent to some vertex $u \in V \setminus S$ that has exactly d neighbours in $G - S$.

Note that if $d = 0$, then the Degree-d-Modulator problem coincides with the well-known Vertex Cover problem. If S is a *degree-1-modulator*, then $G - S$ is a disjoint collection of isolated vertices and isolated edges. Similarly, if S is a *degree-2-modulator*, then $G - S$ consists of isolated vertices, paths and cycles. The minimization version of Degree-d-Modulator has been studied previously. Here we are interested in the maximization version of Degree-d-Modulator. We now define our problem formally.

MaxMin Degree-d-Modulator
Input: An undirected graph $G = (V, E)$ and an integer $k \in \mathbb{N}$.
Question: Does G have a minimal degree-d-modulator of size at least k?

While the Degree-d-Modulator problem has been studied extensively in the literature from both algorithmic and parameterized complexity perspectives [15, 38, 55, 97, 99], the maximization variant that seeks minimal modulators of large size has received comparatively little attention. Notably, the case $d = 0$ corresponds to the well-known MaxMin Vertex Cover (MMVC) problem, which has been previously explored by Fernau [57], who provided fixed-parameter tractable (FPT) algorithms and kernelization results. In particular, it was shown that MMVC admits a kernel with at most k^2 vertices and can be solved in $\mathcal{O}^*(c^k)$ time [114].

In this work, we initiate a broader investigation into the parameterized complexity of MaxMin Degree-d-Modulator for general values of d. We first establish that the problem admits a polynomial kernel of size $\mathcal{O}(dk^2 + d^2k)$, generalizing the kernelization result known for the case $d = 0$. We also show that MaxMin Degree-d-Modulator is FPT when parameterized by the treewidth tw of the input graph.

4.1 Kernelization Algorithm

Lemma 68. *Let (G, k) be an instance of* MaxMin Degree-d-Modulator. *If G contains a vertex v of degree at least $k + d$, then (G, k) is a yes instance.*

Proof. Observe that $V(G) \setminus \{v\}$ is a degree-d-modulator of graph G, since removing all vertices except v leaves v with degree zero, which is at most d. Therefore, there exists a minimal degree-d-modulator $S \subseteq V(G) \setminus \{v\}$. We claim that S contains at least k neighbours of v. Suppose, for the sake of contradiction, that S contains at most $k - 1$ neighbours of v. Then, in the graph $G - S$, vertex v would have degree at least $d(v) - (k - 1) \geq (k + d) - (k - 1) = (d + 1)$, which contradicts the assumption that S is a degree-d-modulator. Therefore, S must contain at least k neighbours of v, implying that (G, k) is a yes-instance. □

We give a few reduction rules that simplify the input instance.

Reduction Rule 1. If G has an isolated vertex v, delete v from G. The new instance is $(G - v, k)$.

Reduction Rule 2. If G contains a vertex v of degree at most d such that all the neighbours of v are also of degree at most d, then delete v from G. The new instance is $(G - v, k)$.

Lemma 68 allows us to claim the final reduction rule that explicitly bounds the size of the kernel.

Reduction Rule 3. If G has a vertex v of degree at least $k + d$, then conclude that we are dealing with a yes-instance.

It is easy to see that Reduction Rules 1, 2 and 3 are safe and that they produce an equivalent instance. Furthermore, all the reduction rules can be applied in polynomial time. Now we are ready to state the main theorem of this section.

Theorem 69. MaxMin Degree-d-Modulator *admits a kernel of size* $dk^2 + d^2k$.

Proof. Given an input instance (G, k), we greedily find a minimal degree-d-modulator S of G. If $|S| \geq k$, then (G, k) is a yes-instance. Otherwise, we have a minimal degree-d-modulator S of size at most $k - 1$. Let $A \subseteq V \setminus S$ be the set of vertices that are not adjacent to any vertex in S, and let $B \subseteq V \setminus S$ be the set of vertices that are adjacent to at least one vertex in S. Clearly $V \setminus S = A \cup B$. Now, we would like to bound the size of $V - S$ by analyzing the sizes of A and B. Since we cannot apply Reduction Rule 3 anymore, every vertex of G has degree at most $k + d$. Furthermore, each vertex in B has a neighbor in S, and there are at most k vertices in S. Therefore, the total number of vertices in B is at most $(k + d)k$. Because S is a degree-d-modulator, every vertex in A has degree at most d. If there exists a vertex $u \in A$ that does not have any neighbor in B, then by Reduction Rule 2, we can safely delete u. Thus, we can assume that every vertex in A has at least one neighbor in B. Additionally, since every vertex in B has at most d neighbor in A, the total number of vertices in A is at most $d \cdot |B| \leq d \cdot k(k + d)$. □

Using the above theorem, we obtain a polynomial kernel of size dk^2+d^2k, which allow us to design a fixed-parameter tractable (FPT) algorithm for MaxMin Degree-d-Modulator with running time $\mathcal{O}^\star(2^{dk^2+d^2k})$ as follows: For every vertex subset S of size at least k, we test whether S is a minimal degree-d-modulator. If we find such an S, then (G, k) is a yes-instance, otherwise, it is a no-instance. We improve this running time for the cases when $d = 0, 1$ and 2.

4.2 MaxMin Degree-d-Modulator parameterized by Treewidth

Theorem 70. *Given a nice tree decomposition of a graph G with treewidth tw, one can determine the size of maximum minimal degree-d-modulator in time $\mathcal{O}\left(4^{\mathsf{tw}} \cdot (d+2)^{\mathsf{tw}} \cdot 2^{\mathsf{tw}} \cdot (d+2)^{\mathsf{tw}} \cdot n^9\right)$, where n is the number of vertices of G.*

Proof. Let $(T, \{X_t\}_{t \in V(T)})$ be a nice tree decomposition rooted at node r of the input graph G. For any node t in T, let V_t denote the union of all bags in the subtree of T rooted at t, including X_t itself. We denote G_t as the subgraph of G induced by V_t. In our algorithm, we not only distinguish whether a vertex is included in the solution or not, but also, for those included, we further differentiate between those that are adjacent to a vertex of degree exactly d outside the solution and those that have at least $d+1$ neighbors outside the solution. Similarly, among the vertices not included in the solution, we differentiate between those that are of degree exactly d after solution vertices are removed and those that are of degree less than d after solution vertices are removed.

A coloring of the bag X_t is a mapping $f : X_t \to \{r, g, y, b\}$, which assigns one of the four colours to each vertex in the bag. We give the intuition behind the meaning of these four colours.

- **Red**, represented by r. All red vertices must be excluded from the final solution and must have less than d neighbors outside the final solution.
- **Grey**, represented by g. All grey vertices must be excluded from the final solution and must have exactly d neighbors outside the final solution.
- **Yellow**, represented by y. All yellow vertices must be included in the final solution and must have at least one grey neighbour.
- **Blue**, represented by b. All blue vertices must be included in the final solution and must have at least $d+1$ neighbours outside the final solution.

For each node t, we construct a table $dp_t(f, \mathbf{C}, \mathbf{Y}, \mathbf{B}, r_0, r_1, g_0, g_1, y_0, y_1, b_0, b_1) \in \{\text{true}, \text{ false}\}$ where f is a colouring of X_t; and $\mathbf{C}, \mathbf{Y}$ and $\mathbf{B}$ are vectors of length n. Let m be a positive

integer and $m_{\geq}$ be a symbol representing any integer value greater than or equal to m. We define

$$m_{\geq} + 1 = m_{\geq} \text{ and } (m-1) + 1 = m_{\geq}.$$

The ith coordinate of $\mathbf{C}$ and $\mathbf{B}$ takes values from the set $\{0, 1, \ldots, d, (d+1)_{\geq}\}$ if and only if v_i is in X_t; otherwise it takes the symbol $\star$. Similarly, the ith coordinate of $\mathbf{Y}$ takes values from the set $\{0, 1_{\geq}\}$ if and only if v_i is in X_t; otherwise, it takes the symbol $\star$. Let $r_0, r_1, g_0, g_1, y_0, y_1, b_0$ and b_1 be integers taking values in the range 0 to n, inclusive.

We set $dp_t(f, \mathbf{C}, \mathbf{Y}, \mathbf{B}, r_0, r_1, g_0, g_1, y_0, y_1, b_0, b_1)$ =true if and only if there exists a set $X \subseteq V_t$ such that

1. $f^{-1}\{y, b\} = X \cap X_t$,
2. the ith coordinate of vector $\mathbf{C}$ is

$$\mathbf{C}(i) = \begin{cases} l, & \text{if } v_i \in X_t \text{ and } v_i \text{ has } l \text{ (with } l \leq d) \text{ red or grey neighbours in } G_t \\ (d+1)_{\geq}, & \text{if } v_i \in X_t \text{ and } v_i \text{ has at least } d+1 \text{ red or grey neighbours in } G_t \\ \star & \text{otherwise} \end{cases}$$

3. the ith coordinate of vector $\mathbf{Y}$ is

$$\mathbf{Y}(i) = \begin{cases} 0, & \text{if } v_i \in X_t \text{ and } v_i \text{ has no grey neighbours in } G_t \\ 1_{\geq}, & \text{if } v_i \in X_t \text{ and } v_i \text{ has at least one grey neighbour in } G_t \\ \star & \text{otherwise} \end{cases}$$

4. the ith coordinate of vector $\mathbf{B}$ is

$$\mathbf{B}(i) = \begin{cases} l, & \text{if } v_i \in X_t \text{ and } v_i \text{ has } l \text{ } (\leq d) \text{ red neighbours in } G_t \\ (d+1)_{\geq}, & \text{if } v_i \in X_t \text{ and } v_i \text{ has at least } d+1 \text{ red neighbours in } G_t \\ \star & \text{otherwise} \end{cases}$$

5. r_0 is the number of red vertices in G_t.
6. r_1 is the number of red vertices in $G_t - X_t$ that have at most $d-1$ red or grey neighbours.

7. g_0 is the number of grey vertices in G_t.

8. g_1 is the number of grey vertices in $G_t - X_t$ that have exactly d red or grey neighbours.

9. y_0 is the number of yellow vertices in G_t.

10. y_1 is the number of yellow vertices in $G_t - X_t$ that have at least one grey neighbour.

11. b_0 is the number of blue vertices in G_t.

12. b_1 is the number of blue vertices in $G_t - X_t$ that have at least $d+1$ red neighbours.

Now we will show how each entry $dp_t(f, \mathbf{C}, \mathbf{Y}, \mathbf{B}, r_0, r_1, g_0, g_1, y_0, y_1, b_0, b_1)$ can be computed, assuming that the entries corresponding to the children of t have already been computed.

Leaf node: For a leaf node ℓ, we have $X_\ell = \emptyset$. Thus $dp_\ell(f, \mathbf{C}, \mathbf{Y}, \mathbf{B}, r_0, r_1, g_0, g_1, y_0, y_1, b_0, b_1)$ is true if and only if $f = \emptyset, \mathbf{C} = *, \mathbf{Y} = *, \mathbf{B} = *, r_0 = 0, r_1 = 0, g_0 = 0, g_1 = 0, y_0 = 0, y_1 = 0, b_0 = 0,\ b_1 = 0$.

Introduce node: Suppose t is an introduce node with child t' such that $X_t = X_{t'} \cup \{v_j\}$ for some $v_j \notin X_{t'}$. Let f be any coloring of X_t. We consider the following cases:

Case (i): Let $f(v_j) = r$. In this case, $dp_t(f, \mathbf{C}, \mathbf{Y}, \mathbf{B}, r_0, r_1, g_0, g_1, y_0, y_1, b_0, b_1)$ is true if and only if $dp_{t'}(f', \mathbf{C}', \mathbf{Y}', \mathbf{B}', r'_0, r'_1, g'_0, g'_1, y'_0, y'_1, b'_0, b'_1)$ is true, where

1. $f|_{X_t \setminus \{v_j\}} = f'$;

2. $$\mathbf{C}(i) = \begin{cases} \mathbf{C}'(i) + 1, & \text{if } v_i \in N(v_j) \cap X_t \\ l, & \text{if } v_i = v_j \text{ and } v_j \text{ has } l\ (\leq d) \text{ red or grey neighbours in } X_t \\ (d+1)_{\geq}, & \text{if } v_i = v_j \text{ and } v_j \text{ has at least } (d+1) \text{ red or grey neighbours in } X_t \\ \mathbf{C}'(i), & \text{otherwise} \end{cases}$$

3.
$$\mathbf{Y}(i) = \begin{cases} 0, & \text{if } v_i = v_j \text{ and } v_j \text{ has no grey neighbours in } X_t \\ 1_{\geq}, & \text{if } v_i = v_j \text{ and } v_j \text{ has at least one grey neighbour in } X_t \\ \mathbf{Y}'(i), & \text{otherwise} \end{cases}$$

4.
$$\mathbf{B}(i) = \begin{cases} \mathbf{B}'(i) + 1, & \text{if } v_i \in N(v_j) \cap X_t \\ l, & \text{if } v_i = v_j \text{ and } v_j \text{ has } l \ (\leq d) \text{ red neighbours in } X_t \\ (d+1)_{\geq}, & \text{if } v_i = v_j \text{ and } v_j \text{ has at least } d+1 \text{ red neighbours in } X_t \\ \mathbf{B}'(i), & \text{otherwise} \end{cases}$$

5. $r_0 = r'_0 + 1$;
6. $r_1 = r'_1$;
7. $g_0 = g'_0$;
8. $g_1 = g'_1$;
9. $y_0 = y'_0$;
10. $y_1 = y'_1$;
11. $b_0 = b'_0$;
12. $b_1 = b'_1$.

Case (ii): Let $f(v_j) = g$. In this case, $dp_t(f, \mathbf{C}, \mathbf{Y}, \mathbf{B}, r_0, r_1, g_0, g_1, y_0, y_1, b_0, b_1)$ is true if and only if $dp_{t'}(f', \mathbf{C}', \mathbf{Y}', \mathbf{B}', r'_0, r'_1, g'_0, g'_1, y'_0, y'_1, b'_0, b'_1)$ is true, where

1. $f|_{X_t \setminus \{v_j\}} = f'$;
2.
$$\mathbf{C}(i) = \begin{cases} \mathbf{C}'(i) + 1, & \text{if } v_i \in N(v_j) \cap X_t \\ l, & \text{if } v_i = v_j \text{ and } v_j \text{ has } l \ (\leq d) \text{ red or grey neighbours in } X_t \\ (d+1)_{\geq}, & \text{if } v_i = v_j \text{ and } v_j \text{ has at least } (d+1) \text{ red or grey neighbours in } X_t \\ \mathbf{C}'(i), & \text{otherwise} \end{cases}$$

3.
$$\mathbf{Y}(i) = \begin{cases} 1_{\geq}, & \text{if } v_i \in N(v_j) \cap X_t \\ 0, & \text{if } v_i = v_j \text{ and } v_j \text{ has no grey neighbours in } X_t \\ 1_{\geq}, & \text{if } v_i = v_j \text{ and } v_j \text{ has at least one grey neighbour in } X_t \\ \mathbf{Y}'(i), & \text{otherwise} \end{cases}$$

4.
$$\mathbf{B}(i) = \begin{cases} l, & \text{if } v_i = v_j \text{ and } v_j \text{ has } l\ (\leq d) \text{ red neighbours in } X_t \\ (d+1)_{\geq}, & \text{if } v_i = v_j \text{ and } v_j \text{ has at least } d+1 \text{ red neighbours in } X_t \\ \mathbf{B}'(i), & \text{otherwise} \end{cases}$$

5. $r_0 = r'_0$;
6. $r_1 = r'_1$;
7. $g_0 = g'_0 + 1$;
8. $g_1 = g'_1$;
9. $y_0 = y'_0$;
10. $y_1 = y'_1$;
11. $b_0 = b'_0$;
12. $b_1 = b'_1$.

Case (iii): Let $f(v_j) = y$. In this case, $dp_t(f, \mathbf{C}, \mathbf{Y}, \mathbf{B}, r_0, r_1, g_0, g_1, y_0, y_1, b_0, b_1)$ is true if and only if $dp_{t'}(f', \mathbf{C}', \mathbf{Y}', \mathbf{B}', r'_0, r'_1, g'_0, g'_1, y'_0, y'_1, b'_0, b'_1)$ is true, where

1. $f|_{X_t \setminus \{v_j\}} = f'$;
2.
$$\mathbf{C}(i) = \begin{cases} l, & \text{if } v_i = v_j \text{ and } v_j \text{ has } l\ (\leq d) \text{ red or grey neighbours in } X_t \\ (d+1)_{\geq}, & \text{if } v_i = v_j \text{ and } v_j \text{ has at least } (d+1) \text{ red or grey neighbours in } X_t \\ \mathbf{C}'(i), & \text{otherwise} \end{cases}$$

3.
$$\mathbf{Y}(i) = \begin{cases} 0, & \text{if } v_i = v_j \text{ and } v_j \text{ has no grey neighbours in } X_t \\ 1_{\geq}, & \text{if } v_i = v_j \text{ and } v_j \text{ has at least one grey neighbour in } X_t \\ \mathbf{Y}'(i), & \text{otherwise} \end{cases}$$

4.
$$\mathbf{B}(i) = \begin{cases} l, & \text{if } v_i = v_j \text{ and } v_j \text{ has } l\ (\leq d) \text{ red neighbours in } X_t \\ (d+1)_{\geq}, & \text{if } v_i = v_j \text{ and } v_j \text{ has at least d+1 red neighbours in } X_t \\ \mathbf{B}'(i), & \text{otherwise} \end{cases}$$

5. $r_0 = r'_0$;

6. $r_1 = r'_1$;

7. $g_0 = g'_0$;

8. $g_1 = g'_1$;

9. $y_0 = y'_0 + 1$;

10. $y_1 = y'_1$;

11. $b_0 = b'_0$;

12. $b_1 = b'_1$.

Case (iv): Let $f(v_j) = b$. In this case, $dp_t(f, \mathbf{C}, \mathbf{Y}, \mathbf{B}, r_0, r_1, g_0, g_1, y_0, y_1, b_0, b_1)$ is true if and only if $dp_{t'}(f', \mathbf{C}', \mathbf{Y}', \mathbf{B}', r'_0, r'_1, g'_0, g'_1, y'_0, y'_1, b'_0, b'_1)$ is true, where

1. $f|_{X_t \setminus \{v_j\}} = f'$;

2.
$$\mathbf{C}(i) = \begin{cases} l, & \text{if } v_i = v_j \text{ and } v_j \text{ has } l\ (\leq d) \text{ red or grey neighbours in } X_t \\ (d+1)_{\geq}, & \text{if } v_i = v_j \text{ and } v_j \text{ has at least } (d+1) \text{ red or grey neighbours in } X_t \\ \mathbf{C}'(i), & \text{otherwise} \end{cases}$$

3.
$$\mathbf{Y}(i) = \begin{cases} 0, & \text{if } v_i = v_j \text{ and } v_j \text{ has no grey neighbours in } X_t \\ 1_{\geq}, & \text{if } v_i = v_j \text{ and } v_j \text{ has at least one grey neighbour in } X_t \\ \mathbf{Y}'(i), & \text{otherwise} \end{cases}$$

4.
$$\mathbf{B}(i) = \begin{cases} l, & \text{if } v_i = v_j \text{ and } v_j \text{ has } l\ (\leq d) \text{ red neighbours in } X_t \\ (d+1)_{\geq}, & \text{if } v_i = v_j \text{ and } v_j \text{ has at least } d+1 \text{ red neighbours in } X_t \\ \mathbf{B}'(i), & \text{otherwise} \end{cases}$$

5. $r_0 = r'_0$;

6. $r_1 = r'_1$;

7. $g_0 = g'_0$;

8. $g_1 = g'_1$;

9. $y_0 = y'_0$;

10. $y_1 = y'_1$;

11. $b_0 = b'_0 + 1$;

12. $b_1 = b'_1$

It is straightforward to observe that for an introduce node t, $dp_t(f, \mathbf{C}, \mathbf{Y}, \mathbf{B}, r_0, r_1, g_0, g_1, y_0, y_1, b_0, b_1)$ can be computed in $\mathcal{O}(n)$ time.

Forget Node. Suppose t is an forget node with child t' such that $X_t = X_{t'} \backslash \{v_j\}$ for some $v_j \in X_{t'}$. Here $dp_t(f, \mathbf{C}, \mathbf{Y}, \mathbf{B}, r_0, r_1, g_0, g_1, y_0, y_1, b_0, b_1)$ is true if and only if $dp_{t'}(f', \mathbf{C}', \mathbf{Y}', \mathbf{B}', r'_0, r'_1, g'_0, g'_1, y'_0, y'_1, b'_0, b'_1)$ is true, where $f' = f_{v_j \to r}, f_{v_j \to g}, f_{v_j \to y}$ or $f_{v_j \to b}$ and the remaining values are determined according to the behavior of f':

Case (i): When $f' = f_{v_j \to r}$

2.
$$\mathbf{C}(i) = \begin{cases} \star & \text{if } v_i = v_j \\ \mathbf{C}'(i) & \text{otherwise} \end{cases}$$

3.
$$\mathbf{Y}(i) = \begin{cases} \star & \text{if } v_i = v_j \\ \mathbf{Y}'(i) & \text{otherwise} \end{cases}$$

4.
$$\mathbf{B}(i) = \begin{cases} \star & \text{if } v_i = v_j \\ \mathbf{B}'(i) & \text{otherwise} \end{cases}$$

5. $r_0 = r'_0$;

6. $r_1 = \begin{cases} r'_1 + 1, & \text{if } \mathbf{C}'(j) \leq d-1 \\ r'_1, & \text{Otherwise} \end{cases}$

7. $g_0 = g'_0$;

8. $g_1 = g'_1$;

9. $y_0 = y'_0$;

10. $y_1 = y'_1$;

11. $b_0 = b'_0$;

12. $b_1 = b'_1$

Case (ii): When $f' = f_{v_j \to g}$

2.
$$\mathbf{C}(i) = \begin{cases} \star & \text{if } v_i = v_j \\ \mathbf{C}'(i) & \text{otherwise} \end{cases}$$

3.
$$\mathbf{Y}(i) = \begin{cases} \star & \text{if } v_i = v_j \\ \mathbf{Y}'(i) & \text{otherwise} \end{cases}$$

4.
$$\mathbf{B}(i) = \begin{cases} \star & \text{if } v_i = v_j \\ \mathbf{B}'(i) & \text{otherwise} \end{cases}$$

5. $r_0 = r'_0$;

6. $r_1 = r'_1$;

7. $g_0 = g'_0$;

8. $g_1 = \begin{cases} g'_1 + 1, & \text{if } \mathbf{C}'(j) = d \\ g'_1, & \text{Otherwise} \end{cases}$

9. $y_0 = y'_0$;

10. $y_1 = y'_1$;

11. $b_0 = b'_0$;

12. $b_1 = b'_1$

Case (iii): When $f' = f_{v_j \to y}$

2.
$$\mathbf{C}(i) = \begin{cases} \star & \text{if } v_i = v_j \\ \mathbf{C}'(i) & \text{otherwise} \end{cases}$$

3.
$$\mathbf{Y}(i) = \begin{cases} \star & \text{if } v_i = v_j \\ \mathbf{Y}'(i) & \text{otherwise} \end{cases}$$

4.
$$\mathbf{B}(i) = \begin{cases} \star & \text{if } v_i = v_j \\ \mathbf{B}'(i) & \text{otherwise} \end{cases}$$

5. $r_0 = r'_0$;

6. $r_1 = r'_1$;

7. $g_0 = g'_0$;

8. $g_1 = g'_1$;

9. $y_0 = y'_0$;

10. $y_1 = \begin{cases} y'_1 + 1, & \text{if } \mathbf{Y}'(j) = 1_{\geq} \\ y'_1, & \text{Otherwise} \end{cases}$

11. $b_0 = b'_0$;

12. $b_1 = b'_1$

Case (iv): When $f' = f_{v_j \to b}$

2. $$\mathbf{C}(i) = \begin{cases} \star & \text{if } v_i = v_j \\ \mathbf{C}'(i) & \text{otherwise} \end{cases}$$

3. $$\mathbf{Y}(i) = \begin{cases} \star & \text{if } v_i = v_j \\ \mathbf{Y}'(i) & \text{otherwise} \end{cases}$$

4. $$\mathbf{B}(i) = \begin{cases} \star & \text{if } v_i = v_j \\ \mathbf{B}'(i) & \text{otherwise} \end{cases}$$

5. $r_0 = r'_0$;

6. $r_1 = r'_1$;

7. $g_0 = g'_0$;

8. $g_1 = g'_1$;

9. $y_0 = y'_0$;

10. $y_1 = y'_1$;

11. $b_0 = b'_0$;

12. $b_1 = \begin{cases} b'_1 + 1, & \text{if } \mathbf{B}'(j) = (d+1)_{\geq} \\ b'_1, & \text{Otherwise} \end{cases}$ □

Join Node. Suppose t is a join node with children t_1 and t_2 such that $X_t = X_{t_1} = X_{t_2}$. Then $dp_t(f, \mathbf{C}, \mathbf{Y}, \mathbf{B}, r_0, r_1, g_0, g_1, y_0, y_1, b_0, b_1)$ is true if and only if $dp_{t_1}(f_1, \mathbf{C}_1, \mathbf{Y}_1, \mathbf{B}_1, r_0^1, r_1^1, g_0^1, g_1^1, y_0^1, y_1^1, b_0^1, b_1^1)$ and $dp_{t_2}(f_2, \mathbf{C}_2, \mathbf{Y}_2, \mathbf{B}_2, r_0^2, r_1^2, g_0^2, g_1^2, y_0^2, y_1^2, b_0^2, b_1^2)$ are true, where

1. $f = f_1 = f_2$
2. $$\mathbf{C}(i) = \begin{cases} \star & \text{if } v_i \notin X_t \\ (d+1)_{\geq}, & \text{if } v_i \in X_t \wedge ((\mathbf{C}_1(i) = (d+1)_{\geq}) \vee (\mathbf{C}_2(i) = (d+1)_{\geq})) \\ l_C & \text{Otherwise.} \end{cases}$$
   where
   $$l_C = \begin{cases} l & \text{if } \mathbf{C}_1(i) + \mathbf{C}_2(i) - |N(v_i) \cap (f^{-1}(r) \cup f^{-1}(g)) \cap X_t| = l \text{ and } l \leq d \\ (d+1)_{\geq}, & \text{if } \mathbf{C}_1(i) + \mathbf{C}_2(i) - |N(v_i) \cap (f^{-1}(r) \cup f^{-1}(g)) \cap X_t| > d \end{cases}$$
3. $$\mathbf{Y}(i) = \begin{cases} \star & \text{if } v_i \notin X_t \\ 0, & \text{if } v_i \in X_t \text{ and } \mathbf{Y}_1(i) = \mathbf{Y}_2(i) = 0 \\ 1_{\geq}, & \text{if } v_i \in X_t \wedge ((\mathbf{Y}_1(i) = 1_{\geq}) \vee (\mathbf{Y}_2(i) = 1_{\geq})) \end{cases}$$
4. $$\mathbf{B}(i) = \begin{cases} \star & \text{if } v_i \notin X_t \\ (d+1)_{\geq}, & \text{if } v_i \in X_t \wedge ((\mathbf{B}_1(i) = (d+1)_{\geq}) \vee (\mathbf{B}_2(i) = (d+1)_{\geq})) \\ l_B & \text{Otherwise.} \end{cases}$$
   where
   $$l_B = \begin{cases} l & \text{if } \mathbf{B}_1(i) + \mathbf{B}_2(i) - |N(v_i) \cap f^{-1}(r) \cap X_t| = l \text{ and } l \leq d \\ (d+1)_{\geq}, & \text{if } \mathbf{B}_1(i) + \mathbf{B}_2(i) - |N(v_i) \cap f^{-1}(r) \cap X_t| > d \end{cases}$$

5. $r_0 = r_0^1 + r_0^2 - |f^{-1}(r)|$
6. $r_1 = r_1^1 + r_1^2$
7. $g_0 = g_0^1 + g_0^2 - |f^{-1}(g)|$
8. $g_1 = g_1^1 + g_1^2$
9. $y_0 = y_0^1 + y_0^2 - |f^{-1}(y)|$
10. $y_1 = y_1^1 + y_1^2$
11. $b_0 = b_0^1 + b_0^2 - |f^{-1}(b)|$
12. $b_1 = b_1^1 + b_1^2$

The number of nodes in the tree decomposition is $\mathcal{O}(\mathsf{tw} \cdot n)$. If the treewidth of the graph is tw then the number of possible tuples is $\mathcal{O}\left(4^{\mathsf{tw}} \cdot (d+2)^{\mathsf{tw}} \cdot 2^{\mathsf{tw}} \cdot (d+2)^{\mathsf{tw}} \cdot n^8\right)$. Each entry $dp_t(f, \mathbf{C}, \mathbf{Y}, \mathbf{B}, r_0, r_1, g_0, g_1, y_0, y_1, b_0, b_1)$ can be computed in $O(1)$ time, assuming that the entries corresponding to the children of t have already been computed. The correctness of the recurrence relations follows directly from the definitions and case analysis for each node type.

At the root node r, we look at all records such that $dp_r(\emptyset, *, *, *, r_0, r_1, g_0, g_1, y_0, y_1, b_0, b_1)=$ true, $r_0 = r_1, g_0 = g_1, y_0 = y_1$ and $b_0 = b_1$. Here $*$ is a function such that $*(v) = \star \forall v \in V(G)$. The size of a maximum minimal degree-d-modulator is the maximum $r_1 + g_1$ satisfying $dp_r(\emptyset, *, *, *, r_0, r_1, g_0, g_1, y_0, y_1, b_0, b_1)=$ true, $r_0 = r_1, g_0 = g_1, y_0 = y_1$ and $b_0 = b_1$.

□

By applying Theorem 70, we show that the MaxMin Degree-d-Modulator can be solved in time $\mathcal{O}^\star(c^k)$ for $d = 0, 1$, and 2. We begin by using a greedy algorithm to compute a minimal degree-d modulator X of the input graph G. If $|X| \geq k$, we immediately return a yes-instance. Otherwise, we may assume that $|X| < k$.

Observe that for $d \in \{0, 1, 2\}$, given a degree-d modulator of size at most k, one can construct a tree decomposition of G with treewidth at most $k + 2$. Therefore, we conclude that $\mathsf{tw}(G) = \mathcal{O}(k)$. Then, by invoking the algorithm from Theorem 70, we can solve the MaxMin Degree-d-Modulator in $\mathcal{O}^\star(c^k)$ time.

4.3 Conclusions and Open Problems

In this chapter, we investigated the parameterized complexity of the MaxMin Degree-d-Modulator. We presented both kernelization and fixed-parameter tractability results for fixed values of d. We first established that MaxMin Degree-d-Modulator admits a polynomial kernel of size $\mathcal{O}(dk^2 + d^2k)$. We developed a dynamic programming algorithm based on tree decompositions to solve the problem. By invoking our treewidth-based algorithm, we showed that the problem is fixed-parameter tractable (FPT) and can be solved in time $\mathcal{O}^*(c^k)$ for some constant c when $d \leq 2$.

The problem of designing an $\mathcal{O}^*(c^k)$-time FPT algorithm for $d \geq 3$ remains open. Moreover, for $d \in \{1, 2\}$, although we obtain an $\mathcal{O}^*(c^k)$-time algorithm, the constant c is quite large. Thus, it would be interesting to investigate whether smaller constants can be achieved.

Chapter 5

Maximum Minimal Feedback Vertex Set

In Feedback Vertex Set (FVS), we are given an undirected graph G, a positive integer k, and the question is to check whether G has a vertex set S of size at most k such that $G-S$ is a forest (an acyclic graph). The set S is called *feedback vertex set* or *fvs* in short. Downey and Fellows [45], and Bodlaender [22] proposed the first two fixed parameter tractable (FPT) algorithms with the running time $\mathcal{O}^\star(2^{\mathcal{O}(k \log k)})$[1]. After a long series of improvements, the current champion algorithms are as follows. The fastest known randomized algorithm is given by Li and Nederlof [89] and runs in time $\mathcal{O}^\star(2.7^k)$, and the fastest known deterministic algorithm, given by Iwata and Kobayashi [80], runs in time $\mathcal{O}^\star(3.460^k)$. Several minimization variants of FVS have been studied in the literature, such as finding a set S such that $G - S$ is acyclic and $G[S]$ is connected, or $G[S]$ is an independent set. In this chapter, we consider a maximization version of FVS, namely MaxMin FVS. A set S is called a *minimal fvs*, if S is an fvs and for every $v \in S$, $S \setminus \{v\}$ is not an fvs. That is, no proper subset of S is an fvs. It is not hard to see that if S is a minimal fvs, then every $u \in S$ has a *private cycle*, that is, there exists a cycle in $G[(V(G) \setminus S) \cup \{u\}]$, which goes through u. Now we are ready to define the problem formally.

MaxMin Feedback Vertex Set (MaxMin FVS)
Input: An undirected graph $G = (V, E)$ and an integer $k \in \mathbb{N}$.
Question: Does G admit a minimal fvs of size greater than or equal to k?

[1] The notation $\mathcal{O}^\star$ hides the polynomial factor in the running time.

Lately, finding a large-size minimal solution has attracted a lot of attention from the perspective of the Approximation Algorithms and Parameterized Complexity. Boria et al. [24] proved that for any constant $\epsilon > 0$, the optimization version of MaxMin Vertex Cover is inapproximable within ratios $\mathcal{O}(n^{0.5-\epsilon})$, unless P=NP. They complement this result by proving that MaxMin Vertex Cover is approximable within ratio $\mathcal{O}(n^{0.5})$ in polynomial time. This is in sharp contrast to the approximability of the classical Vertex Cover, for which an easy factor 2-approximation exists. This becomes even more interesting when we consider the optimization version of MaxMin FVS. MaxMin FVS was first considered by Mishra and Sikdar [94], who showed that the problem does not admit an $n^{0.5-\epsilon}$ approximation (unless P=NP), and that it remains APX-hard even when the input graph is of degree at most 9. Dublois et al. [48] improved upon this by showing the first non-trivial polynomial time approximation for MaxMin FVS with a ratio of $\mathcal{O}(n^{\frac{2}{3}})$, as well as a matching hardness of approximation bound of $n^{\frac{2}{3}-\epsilon}$.

In the realm of parameterized complexity, Zehavi [114] studied MaxMin Vertex Cover – find a *minimal vertex cover* of size at least k, if it exists – and designed an algorithm with running time $\mathcal{O}^\star(2^{\mathsf{vc}(G)})$, which is "almost optimal" unless Strong Exponential Time Hypothesis fails. For MaxMin FVS, Dublois et al. [48] obtained a polynomial kernel of size $\mathcal{O}(k^3)$. This result is the starting point of our work. There are results about kernelization of MaxMin Vertex Cover and MaxMin FVS in [7]. In particular, they proved that MaxMin VC parameterized by vertex cover number does not admit a polynomial kernel. This result is related to the kernelization of MaxMin FVS parameterized by vertex cover of our work. This chapter is based on the article [67].

5.1 FPT algorithm parameterized by vertex cover number

In this section we prove that MaxMin FVS is FPT when parameterized by vertex cover number $\mathsf{vc}(G)$.

Theorem 71. MaxMin FVS *can be solved in time* $\mathcal{O}^\star(2^{\mathcal{O}(\mathsf{vc}(G)\log \mathsf{vc}(G))})$.

Proof. If $G = (V, E)$ has a vertex v of degree at most 1, remove it from the graph. We find

a vertex cover C of size at most $\mathsf{vc}(G)$ of the reduced graph G. For our purpose, a standard branching algorithm with $O(2^{\mathsf{vc}(G)}n)$ running time is sufficient (see e.g. [40]). We denote by I the independent set $V \setminus C$. We next guess $C_{in} = S \cap C$, where S is a largest minimal fvs. There are at most $2^{\mathsf{vc}(G)}$ candidates for C_{in} as each member of C has two options: either in $C_{in} = S \cap C$ or $C_{out} = \bar{S} \cap C$. Clearly, $C_{out} = \bar{S} \cap C$ contains the vertices of C which are outside the solution. If $G[C_{out}]$ is not a forest then return that C_{in} is a wrong guess, and we reject the guess. So, from now on, we assume that $G[C_{out}]$ is indeed a forest. Next, we check the minimality of C_{in}, that is, for each v in C_{in}, whether $G[\big(V \setminus C_{in}\big) \cup \{v\}]$ has a cycle containing v. If the minimality of C_{in} is not satisfied, then return that C_{in} is a wrong guess.

Outline of the algorithm: Given a guess C_{in}, our goal is to find a largest minimal fvs containing C_{in}. We look for a set $Z \subseteq I$ of vertices which can be added to C_{out} so that $G[C_{out} \cup Z]$ remains a forest and every vertex $v \in I \setminus Z$ has at least two neighbours in some component of $G[C_{out} \cup Z]$. This is why every vertex in $I \setminus Z$ must be included in the solution. Finally the algorithm outputs $C_{in} \cup (I \setminus Z)$.

Algorithm to find Z: If $G - C_{in}$ has a vertex v of degree at most 1, remove it from the graph. If there is a vertex v in I such that $G[C_{out} \cup \{v\}]$ contains a cycle, then include v in the solution. Suppose S' is a minimal fvs such that $C_{in} \subseteq S'$. We know $G - S'$ is a forest $\mathcal{F}$. Suppose $\mathcal{F}$ has exactly q trees $T_1, T_2, \ldots, T_q$. Note that the number of trees in $\mathcal{F}$ is at most $\mathsf{vc}(G)$, that is, $q \leq \mathsf{vc}(G)$. We guess a partition $\mathcal{P} = \{P_1, P_2, \ldots, P_q\}$ of C_{out}. We say the partition $\mathcal{P} = \{P_1, P_2, \ldots, P_q\}$ corresponds to trees of $\mathcal{F}$, if $P_i = C_{out} \cap V(T_i)$. For each part P_i there must exist a set $Z_i \subseteq I$ of vertices such that $G[P_i \cup Z_i] = T_i$. Otherwise $\mathcal{P}$ is a wrong guess. Note that $Z = \bigcup_{i=1}^{q} Z_i$. There are at most $\mathsf{vc}(G)^{\mathsf{vc}(G)}$ candidates for $\mathcal{P}$ and we try out all guesses.

Algorithm to find Z_i: Consider ith part P_i of $\mathcal{P}$. Note that $G[P_i]$ is a collection of trees. Given P_i, we want to have a set $Z_i \subseteq I$ of vertices such that $G[P_i \cup Z_i]$ is a tree. This can happen in different ways. For example, it may be the case that only one vertex z of I connects all trees of $G[P_i]$ to form a single tree. It may be the case that we need $s_i > 1$ vertices of I to connect all trees of $G[P_i]$ to form a single tree. We further guess a partition $\mathcal{P}_i = \{P_{i1}, P_{i2}, \ldots, P_{is_i}\}$ of P_i into s_i parts. For each P_i there are at most $\mathsf{vc}(G)^{|P_i|}$

possible partitions. Given a partition $\mathcal{P}_i = \{P_{i1}, P_{i2}, \ldots, P_{is_i}\}$ of P_i we want to have a set $Z_i = \{z_{i1}, z_{i2}, \ldots, z_{is_i}\}$ of vertices such that $z_{ij} \in I$ is adjacent to exactly one vertex of every tree in P_{ij}. Thus, $G[z_{ij} \cup P_{ij}]$ forms a tree. Next, we guess how these s_i trees are joined to form a single tree. We need $s_i - 1$ *cross edges* of the form (z_{ij}, v_{ik}) where $v_{ik} \in P_{ik}$, $j \neq k$ to join s_i trees (See Figure 5.1). There are $s_i(s_i - 1)$ cross edges of the form (z_{ij}, v_{ik}) where

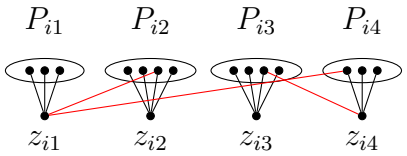
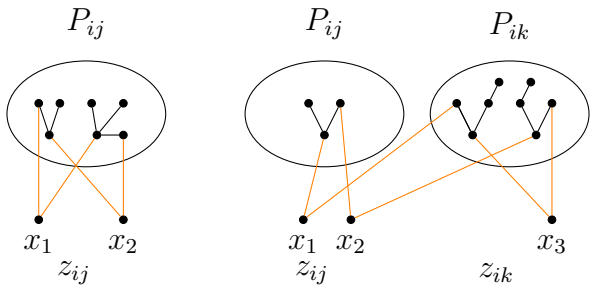


Figure 5.1: An illustration for partition of P_i into four parts $P_{i1}, P_{i2}, P_{i3}, P_{i4}$; cross edges $(z_{i1}, v_{i2}), (z_{i1}, v_{i4}), (z_{i4}, v_{i3})$ are shown in orange. For simplicity trees in each part are singleton.

$v_{ik} \in P_{ik}$, $j \neq k$. Thus $s_i - 1$ edges can be selected in at most $(s_i^2)^{s_i}$ many ways. So the total number of selections of cross edges for all P_i's together is at most $\prod_{i=1}^{q} (s_i^2)^{s_i} \leq \mathsf{vc}(G)^{2\mathsf{vc}(G)}$. Thus there are total $\mathsf{vc}(G)^{4\mathsf{vc}(G)}$ guesses for the structure of trees involving vertices $V(P_i) \cup Z_i$ and cross edges for all i combined. A selection of $s_i - 1$ cross edges is a valid selection if the chosen cross edges can join s_i trees to form one tree. For example, three cross edges (orange) in Figure 5.1 can join four trees to produce one tree; so these three orange edges form a valid selection. Note that the vertices in Z_i are abstract; now we look for candidates of $z_{ij} \in Z_i$ for all j, in the input graph. For a given guess, a vertex $x \in I$ is a *candidate* for z_{ij} if and only if x is only adjacent to exactly one vertex from each tree in P_{ij} and also it is adjacent to a vertex in P_{ik} when (z_{ij}, v_{ik}) is a cross edge in the given guess. For a particular guess there could be several candidates for z_{ij}. We claim that only one of the candidates of z_{ij} can go to Z_i and the rest go to the solution. Suppose there are two candidates x_1 and x_2 for z_{ij}. Without loss of generality assume that x_1 goes to Z_i, that is, x_1 does not go to the solution. Then we prove that x_2 must go to the solution. We consider two cases:

Case 1: Assume that P_{ij} contains at least two trees. Then there is a cycle C containing x_2 where the remaining vertices of C are outside the solution; see Figure 5.2 (a). Therefore x_2 must go to the solution in order to destroy C.

Case 2: Assume that P_{ij} contains exactly one tree. Recall that if $G - C_{in}$ has a vertex v of degree at most 1, then we remove it from the graph. Therefore x_1 and x_2 have



(a) Case 1 (b) Case 2

Figure 5.2: Illustration of Case 1 and Case 2.

degree at least two. Since $d(x_1), d(x_2) \geq 2$, both of them must have a neighbour in some P_{ik} where $j \neq k$, and (z_{ij}, v_{ik}) is a cross edge in the given guess. Therefore, the graph $G[V(P_{ij}) \cup V(P_{ik}) \cup \{x_1, x_2, x_3\}]$ has a cycle C containing x_2 where x_3 is a candidate vertex for z_{ik} and x_3 is outside the solution (See Figure 5.2 (b)). Note that C contains x_2 where the remaining vertices are outside the solution. So x_2 must be inside the solution in order to destroy C; this proves the claim.

So while choosing a candidate for z_{ij}, we only make sure that the remaining candidates which are going to the solution do not disturb the minimality of C_{in}. If there is no such candidate then we return that it is a wrong guess. Clearly, it takes polynomial time to check if Z_i exists for all i. For a given guess C_{in}, if $Z = \bigcup_{i=1}^{q} Z_i$ exists in I then we see that $C_{in} \cup (I \setminus Z)$ forms a minimal feedback vertex set containing C_{in}.

Given C_{in}, in order to compute Z we consider at most $\mathsf{vc}(G)^{4\mathsf{vc}(G)}$ guesses. Thus given C_{in} we can compute Z in time $\mathsf{vc}(G)^{4\mathsf{vc}(G)} n^{O(1)}$. Given C_{in} the above algorithm returns either a minimal fvs containing C_{in} or returns C_{in} is a wrong guess. Finally we consider the maximum size solution obtained over all guesses. As there are $2^{\mathsf{vc}(G)}$ candidates for C_{in}, we can solve the problem in $\mathsf{vc}(G)^{5\mathsf{vc}(G)} n^{O(1)}$ time.

□

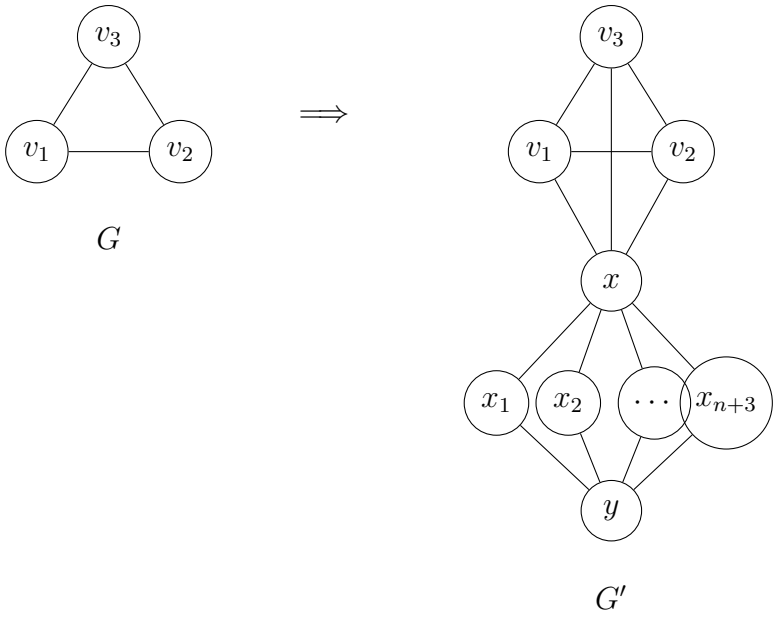


Figure 5.3: Illustration of the PPT reduction from MaxMin Vertex Cover to MaxMin Feedback Vertex Set.

5.2 No polynomial kernel parameterized by $\mathsf{vc}(G)$

We proved that MaxMin FVS is FPT when parameterized by vertex cover number $\mathsf{vc}(G)$, and in this section we show kernelization hardness of the problem.

Theorem 72. MaxMin FVS *parameterized by* $\mathsf{vc}(G)$ *does not admit a polynomial compression unless coNP* $\subseteq$ *NP/poly.*

Proof. We give a polynomial parameter transformation (PPT) from the MaxMin Vertex Cover. Given an instance (G, k) of MaxMin Vertex Cover, we construct in polynomial time an instance (G', k') of MaxMin FVS as follows. We start with the graph G. We add a new vertex x and make it adjacent to every vertex of G. We add a set $V_x = \{x_1, x_2, \ldots, x_{n+3}\}$ of new vertices and make x adjacent to every vertex of V_x. Furthermore, we add a new vertex y and make y adjacent to every vertex of V_x. This completes the construction of G' (See Figure 5.3). It is easy to see that $\mathsf{vc}(G') \leq \mathsf{vc}(G) + 2$. Finally, we set $k' = k + n + 2$. We claim that G contains a minimal vertex cover of size at least k if and only if G' contains a minimal feedback vertex set of size at least k'.

Suppose C is a minimal vertex cover in G such that $|C| \geq k$. We observe that the set

$C \cup (V_x \setminus \{x_1\})$ forms a minimal feedback vertex set of size at least k' in G'. Conversely, suppose that G' has a minimal feedback vertex set S of size at least k'. First, we see that $x \notin S$. This is true because if $x \in S$ then $(V_x \cup \{y\}) \cap S = \emptyset$ as the vertices in $V_x \cup \{y\}$ are not part of any cycle in $G' - x$. This implies that $|S| \leq n+1$, which is a contradiction. Similarly, we can argue that $y \notin S$. As we know that $\{x, y\} \cap S = \emptyset$, we must have $|S \cap V_x| = n + 2$. This implies that $|S \cap V(G)| \geq k$. Now, we show that $C = S \cap V(G)$ is a minimal vertex cover of G. It is easy to see that $C = S \cap V(G)$ is a vertex cover of G; otherwise, S will not be a feedback vertex set of G'. For the sake of contradiction, assume that $C \setminus \{x\}$ is also a vertex cover of G. In this case, we observe that x is not part of any cycle in $G' - (S \setminus \{x\})$, which is a contradiction. Therefore, C is a minimal vertex cover of size at least k in G. □

5.3 An fpt-AS for MaxMin FVS parameterized by $\mathsf{vc}(G)$

An *fpt-approximation scheme (fpt-AS)* with parameterization κ is an algorithm whose input is an instance $x \in I$ and an $\epsilon > 0$, and it produces a $(1-\epsilon)$-approximate solution in time $f(\epsilon, \kappa(x)) \cdot |x|^{\mathcal{O}(1)}$ for some computable function f.

Theorem 73. *Let $\epsilon > 0$ be a fixed constant. Then, there exists an algorithm for* MaxMin FVS*, that runs in time $2^{\mathcal{O}\left(\frac{\mathsf{vc}(G)}{\epsilon}\right)} \cdot n^{\mathcal{O}(1)}$ and returns a minimal feedback vertex set of size at least $(1-\epsilon)\mathsf{opt}(G)$.*

Proof. We present an $f(\epsilon, \mathsf{vc}(G)) \cdot n^{O(1)}$ time algorithm that produces a $(1-\epsilon)$-approximate solution for the problem. We assume that we have a minimum vertex cover C of size $\mathsf{vc}(G)$ of the input graph $G = (V, E)$. We denote by I the independent set $V \setminus C$. Our goal here is to find a largest minimal fvs S with $C_{in} = S \cap C$, where $C_{in} \subseteq C$ is given. That is, we guess the intersection of S with vertex cover C. There are $2^{\mathsf{vc}(G)}$ possible guesses. Clearly, $C_{out} = \bar{S} \cap C$ contains the vertices of C which are outside the solution. If $G[C_{out}]$ is not a forest then return that C_{in} is a wrong guess and reject it. So from now onwards we assume that $G[C_{out}]$ is indeed a forest. We give some reduction rules to simplify the input instance.

Reduction MMFVS 1. If there is a vertex $u \in I$ with at most one neighbour in C_{out}, delete u.

Reduction MMFVS 2. If there is a vertex $u \in I$ such that $G[C_{out} \cup \{u\}]$ contains a cycle, then include u in the solution and delete u.

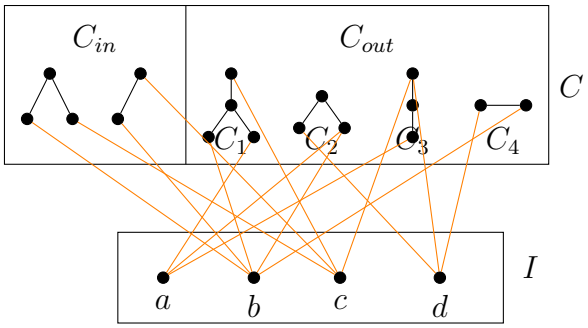


Figure 5.4: Here $Q(a) = \{C_1, C_2, C_3\}$, $Q(b) = \{C_1, C_2, C_4\}$, $Q(c) = \{C_1, C_3\}$, $Q(d) = \{C_2, C_3, C_4\}$. Note that $S_a = \{b, c, d\}$.

The algorithm first applies Reductions MMFVS 1 and MMFVS 2 exhaustively. Every vertex $u \in I$ has at least two neighbours in C_{out}, otherwise Reduction MMFVS 1 would have been applied. Since Reduction MMFVS 2 cannot be applied, we have that no two neighbours of $u \in I$ belong to the same connected component of $G[C_{out}]$. On the reduced instance, we run the following greedy algorithm. Suppose that C_{out} has connected components $C_1, C_2, \ldots$. We say a connected component C_i is a neighbour of $u \in I$, that is $C_i \in Q(u)$, if G contains an edge (u, v) for some $v \in C_i$. pick an arbitrary vertex $u \in I$ and define $S_u = \Big\{x_i \in I \setminus \{u\} \ : \ |Q(u) \cap Q(x_i)| \geq 2\Big\}$. Note that if u is not included in S then all the vertices of S_u must be included in S. The intention is that if $|S_u| \geq 2$, then we prefer not to include u in S, and hence include all the vertices of S_u in S as it is a maximization problem. But while including the vertices of S_u in the solution, we need to be careful about whether the inclusion of S_u in the solution, disturbs the minimality property of C_{in}. This can be verified by checking if each vertex in C_{in} still has a private cycle after the inclusion of S_u in the solution. Based on the above observations, we propose the following algorithm. We pick an arbitrary vertex $u \in I$, compute S_u and check the minimality of C_{in}, assuming S_u is included in the solution. If the minimality of C_{in} is preserved, then we set $S = S \cup S_u$, $I = I \setminus (S_u \cup \{u\})$, and $C_{out} = C_{out} \cup \{u\}$, that is, we include S_u in the solution and move u to C_{out}. As u has neighbours in at least two connected components of $G[C_{out}]$, when we move u to C_{out}, the number of components in $G[C_{out}]$ drops by at least 1. The algorithm again applies Reductions MMFVS 1 and MMFVS 2 exhaustively as C_{out} has been modified. On the other hand, if the minimality of C_{in} is not preserved then we set $S = S \cup \{u\}$ and $I = I \setminus \{u\}$. We repeat the above until I becomes empty.

There are $2^{\mathsf{vc}(G)}$ candidates for C_{in}; for each guess the above algorithm returns a minimal

fvs and finally, we consider the maximum size solution obtained over all guesses. Suppose the algorithm outputs S. Let S_{opt} be an optimum solution. We claim that $|S| \geq |S_{opt}| - \mathsf{vc}(G)$. Let $C_{in} = C \cap S_{opt}$. Clearly, we have $|S_{opt}| \leq |C_{in}| + |I|$. Recall that the greedy algorithm adds all vertices from I to the solution except the ones that are moved to C_{out}. We claim that there are at most $\mathsf{vc}(G)$ vertices from I that are moved to C_{out} and therefore not added to the solution. Every time a vertex is moved to C_{out}, the number of connected components in $G[C_{out}]$ drops by at least one. After moving $\mathsf{vc}(G) - 1$ vertices, the number of connected components in $G[C_{out}]$ becomes one. Therefore, we have moved at most $\mathsf{vc}(G)$ vertices to C_{out}. This proves that $|S| \geq |C_{in}| + I - \mathsf{vc}(G) \geq |S_{opt}| - \mathsf{vc}(G)$.

Next, we propose an FPT approximation scheme. Given an input graph G, we first ask whether G has a minimal fvs of size at least $\frac{\mathsf{vc}(G)}{\epsilon}$. Note that this can be answered in time $(9.34)^{\frac{\mathsf{vc}(G)}{\epsilon}} \cdot n^{\mathcal{O}(1)}$ using the FPT algorithm proposed in [87]. If this is a no-instance, then we can find $\mathsf{opt}(G)$ in the same time by repetitively using the FPT algorithm. If this is a yes-instance, then obviously $\mathsf{opt}(G) \geq \frac{\mathsf{vc}(G)}{\epsilon}$. Hence the value of the constructed solution S is at least

$$\frac{\mathsf{opt}(G) - \mathsf{vc}(G)}{\mathsf{opt}(G)} \geq 1 - \frac{\mathsf{vc}(G)}{\mathsf{opt}(G)} \geq 1 - \epsilon$$

times the optimum. That is, the constructed solution S is a $(1-\epsilon)$-approximate solution. □

5.4 Conclusions and Open Problems

In this chapter, we have studied MaxMin FVS parameterized by the vertex cover number of the input graph. We gave a $\mathcal{O}^\star(2^{\mathcal{O}(\mathsf{vc}(G)\log \mathsf{vc}(G))})$ time algorithm parameterized by $\mathsf{vc}(G)$. Finally, we proposed an fpt-AS parameterized by $\mathsf{vc}(G)$ with better running time $2^{\mathcal{O}\left(\frac{\mathsf{vc}(G)}{\epsilon}\right)} n^{\mathcal{O}(1)}$. We list some nice problems that emerge from the results here: can our algorithm parameterized by $\mathsf{vc}(G)$ be made $c^{\mathcal{O}(\mathsf{vc}(G))}$ time algorithm, for a fixed constant c? It would be interesting to see if the idea in Theorem 71 can be extended to study MaxMin FVS parameterized by $\mathsf{fvs}(G)$.

Chapter 6

MaxMin Separation with Uncut Constraints

In this chapter, we investigate the parameterized complexity of three MaxMin separation problems: Maximum Minimal Multiway Cut Uncut (MaxMin Multiway Cut Uncut), Maximum Minimal List Allocation (MaxMin List Allocation) and Maximum Minimal Subset Feedback Vertex Set (MaxMin Subset FVS). A notable aspect of these problems is that they generalize some of the well-known classical MaxMin graph-cut problems, including MaxMin st-Separator, MaxMin Multiway Cut, MaxMin Feedback Vertex set and Maximum Cut.

In the MaxMin Multiway Cut-Uncut problem, the input is an undirected graph G, a set $T \subseteq V(G)$ of terminals together with a partition $\mathcal{T} = (T_1, \ldots, T_p)$ of T, and a positive integer k. The goal is to determine if there exists a minimal set Z of edges of size at least k with property that, in the graph $G - Z$ all the terminals in T_i are contained in a single connected component for each $i \in [p]$, and for each $i, j \in [p]$, $i \neq j$, the connected component containing T_i is distinct from that containing T_j.

In the MaxMin Subset FVS problem, the input is an undirected graph G, a set of terminals $T \subseteq V(G)$ and a positive integer k. The goal is to determine if there exists a minimal set $X \subseteq V(G)$ of size at least k with the property that, $G - X$ do not contain a cycle intersecting the vertices in T.

Finally, in the MaxMin List Allocation problem, the input is an undirected graph G, a positive integer r, a set of non-negative integers α_{ij} for $1 \leq i < j \leq r$, and a function $\lambda : V(G) \to 2^{[r]}$, we want to determine if there exists a partition of $V(G)$ into r non-empty parts $V_1, \ldots, V_r$ such that $|E(V_i, V_j)| \geq \alpha_{ij}$ for all $1 \leq i < j \leq r$ and for all $v \in V_l$ implies $l \in \lambda(v)$.

Here we formally define the problems studied in this chapter.

MaxMin Multiway Cut-Uncut
Input: An undirected graph G, a set $T \subseteq V(G)$ of terminals together with a partition $(T_1, \ldots, T_p)$ of T, and a positive integer k.
Question: Does there exist a set $X \subseteq E(G)$ of size at least k, such that X is minimal and in $G - X$, each T_i is entirely contained in a unique connected component?

MaxMin Separator-Unseparator
Input: An undirected graph G, a set $T \subseteq V(G)$ of terminals together with a partition $(T_1, \ldots, T_p)$ of T, and a positive integer k.
Question: Does there exist a set $S \subseteq V(G) \setminus T$ of size at least k, such that S is minimal and in $G - S$, each T_i is contained in a unique connected component?

MaxMin Subset Feedback Vertex Set
Input: An undirected graph G, a set $T \subseteq V(G)$ and a positive integer k.
Question: Does there exist $X \subseteq V(G)$ of size at least k, such that X is a minimal set with the property that in $G - X$ there is no cycle that intersects the set T?

MaxMin List Allocation
Input: An undirected graph G, a function $\lambda : V(G) \to 2^{[r]}$, a set of non-negative integers α_{ij} and a positive integer r.
Question: Does there exists a partition of $V(G)$ into r non-empty parts $V_1, V_2, \ldots, V_r$ such that $|E(V_i, V_j)| \geq \alpha_{ij}$ for all $1 \leq i < j \leq r$ and $v \in \lambda(v)$ for all $v \in V(G)$?

Our main contribution is a comprehensive parameterized complexity classification of MaxMin Multiway Cut-Uncut in both edge- and vertex-deletion variants. A key challenge arises from the uncut constraint, which introduces substantial algorithmic difficulties.

Unlike the problems addressed by Gaikwad et al. [66], even finding a feasible solution for Multiway Cut-Uncut is non-trivial and is equivalent to the well-known NP-hard problem of Vertex-Disjoint Paths. A common strategy for tackling MaxMin problems involves greedily constructing a feasible solution and converting it into a minimal one. However, in MaxMin Multiway Cut-Uncut, constructing even this initial feasible solution is highly non-trivial. We present an FPT algorithm that carefully builds a solution while preserving the uncut constraints (details appear in the overview section). In the vertex-deletion setting, the only FPT case corresponds to MaxMin Multiway Separator with undeletable terminals. To prove it is FPT, we reduce it to the more general MaxMin Subset Feedback Vertex Set problem and show that this too is FPT.

Our investigation also includes the MaxMin List Allocation, which generalizes Multiway Cut and was introduced by Kim, Paul, Sau, and Thilikos [85]. In this problem, we are given a graph G, a positive integer r, a set of non-negative integers α_{ij} for $1 \leq i < j \leq r$, and a function $\lambda : V(G) \to 2^{[r]}$. The goal is to partition $V(G)$ into r non-empty parts $V_1, \ldots, V_r$ such that $|E(V_i, V_j)| \geq \alpha_{ij}$ for all $1 \leq i < j \leq r$, and every vertex $v \in V_l$ satisfies $l \in \lambda(v)$. For this problem as well, we establish a complete FPT versus W[1]-hard dichotomy. Interestingly, the FPT algorithm for MaxMin List Allocation turns out to be significantly simpler than those required for the separation problems discussed earlier. Now we will discuss some theory that will be helpful in this chapter.

For two positive integers q and k, a graph G is said to be (q, k)-*vertex unbreakable* if *no* vertex set of size at most k can separate two vertex subsets of size at least q each. We say that a graph is *unbreakable* if it is (q, k)-unbreakable for some values of q and k. In a similar way, one can define the notion of (q, k)-*edge unbreakable* and *unbreakable* graphs. It is not hard to prove that if G is (q, k)-edge unbreakable and G' is obtained from G by subdividing every edge once then G' is (q, k)-vertex unbreakable.

We refer to [40] for the formal definition of Counting Monadic Second Order (CMSO) logic. We will crucially use the following result of Lokshtanov et al. [91] that allows one to show that a CMSO-expressible graph problem is FPT by designing an FPT algorithm for the problem on (q, k)-vertex unbreakable graphs, for any k and a sufficiently large q that depends only on k.

Theorem 74 ([91]). *Let ψ be a CMSO formula. For all $c \in \mathbb{N}$, there exists $s \in \mathbb{N}$ such that if there exists an algorithm that solves CMSO[ψ] on (s, c)-vertex unbreakable structures in*

time $\mathcal{O}(n^d)$ for some $d > 4$, then there exists an algorithm that solves CMSO[ψ] on general structures in time $\mathcal{O}(n^d)$.

Next, we state the Sunflower Lemma which is used in this chapter. We first define the terminology used in the statement of the next lemma. Given a set system $(U, \mathcal{F})$, where U is a set and $\mathcal{F}$ is a family containing distinct subsets of U, a *sunflower* with k petals and a core Y is a collection of sets $S_1, S_2, \ldots, S_k \in \mathcal{F}$ such that $S_i \cap S_j = Y$ for all $i \neq j$. The sets $S_i \setminus Y$ are called the *petals* of this sunflower. If the sets in $\mathcal{F}$ are distinct and $k \geq 2$, then none of the petals of the sunflower are empty. Note that a family of pairwise disjoint sets is a sunflower (with an empty core).

Theorem 75 (Sunflower Lemma, [40, 49])**.** *Let $\mathcal{F}$ be a family of distinct sets over a universe U, such that each set in $\mathcal{F}$ has cardinality exactly d. If $|\mathcal{F}| > d!(k-1)^d$, then $\mathcal{F}$ contains a sunflower with k petals and such a sunflower can be computed in time polynomial in $|\mathcal{F}|$, $|U|$, and k.*

We observe that the MaxMin Multiway Cut-Uncut generalizes the MaxMin Multiway Cut, which in turn generalizes the MaxMin *st*-Cut. It is known that MaxMin *st*-Cut is NP-hard even when restricted to planar bipartite graphs [47]. Since both MaxMin Multiway Cut and MaxMin Multiway Cut-Uncut generalize this problem, it follows that they are also NP-hard on planar bipartite graphs. Similarly, when $r = 2$ and $\lambda(v) = \{1, 2\}$ for all $v \in V(G)$, the MaxMin List Allocation coincides with the classical Maximum Cut, which was shown to be NP-hard even when restricted to graphs of maximum degree 3 by Yannakakis [113]. Hence, MaxMin List Allocation is also NP-hard. Furthermore, the MaxMin Subset FVS generalizes the MaxMin Feedback Vertex Set, which has been shown to be NP-hard on graphs of maximum degree 6 by Dublois et al. [48], who also present an approximation algorithm with ratio $n^{\frac{2}{3}}$ and proved that this is optimal unless P=NP. It follows that MaxMin Subset FVS is NP-hard as well.

We show in Theorem 76 that MaxMin Multiway Cut-Uncut is FPT parameterized by $k + p$ when $|T_i| \leq 2$ for each $i \in [p]$. In this case, the feasibility question (the case of $k = 0$) can be reduced to the Vertex Disjoint Paths, which admits an FPT algorithm parameterized by p [104, 105]. This algorithm forms an important black-box of our algorithm. Since the exponential parts of the running time of this algorithm is tremendous, and we further use the meta-theorem of Lokshtanov et al. [91] (as explained later) which results

in not explicit bounds on the exponential function, the running time of Theorem 76 is not explicitly stated. We list down some of the main theorems proved in this chapter.

Theorem 76. MaxMin Multiway Cut-Uncut *is FPT parameterized by* $p + k$ *when* $|T_i| \leq 2\ \forall i \in [p]$.

As a subroutine, we also construct an FPT algorithm running in time $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(1)}$ for the MaxMin Multiway Cut problem.

Theorem 77. MaxMin Subset FVS *is FPT when parameterized by solution size.*

Theorem 78. *There exists a randomized algorithm that, given an instance of* MaxMin List Allocation, *in time* $r^{\binom{r}{2}\alpha_{\max}}.n^{\mathcal{O}(1)}$ *either reports a failure or finds a* r*-colorful certificate of correctness of the given instance. Moreover, if the algorithm is given a yes instance, it returns a solution with constant probability.*

6.1 Technical Overview

To prove Theorem 76 and Theorem 77, we rely on the result by [91] Lokshtanov et al., who showed that, in order to show fixed-parameter tractability of CMSO definable problems parameterized by the solution size (say k), it is enough to design FPT algorithms for such problems when the input graph is (q, k)-unbreakable, for some large enough q that depends only on k and the problem. Therefore the first step to show that MaxMin Multiway Cut-Uncut and MaxMin Subset FVS are CMSO definable.

In particular, to prove Theorem 76 and Theorem 77, it is enough to prove Theorem 79 and Theorem 80, respectively.

Theorem 79. *For positive integers* $q, k \geq 1$, MaxMin Multiway Cut-Uncut *when* $|T_i| \leq 2$ *for all* $i \in [p]$, *on* (q, k)*-unbreakable graphs on* n *vertices can be solved in time* $f(q, k) \cdot n^{\mathcal{O}(1)}$, *for some function* f *that depends only on* q *and* k.

Theorem 80. *For any positive integers* $q, k \geq 1$, MaxMin Subset FVS *on* (q, k)*-unbreakable graphs on* n *vertices can be solved in time* $2^{(qk)^{\mathcal{O}(q)}} \cdot n^{\mathcal{O}(1)}$.

MaxMin Multiway Cut. We begin by proving that any minimal t_it_j-cut in the graph can be extended to a minimal T-multiway cut. This structural insight allows us to generalize the algorithm proposed by Duarte et al. [47] for the MaxMin st-Cut problem, which they refer to as the Largest st-Bond.

Duarte et al. showed that, given a 2-vertex-connected graph G and terminals $s, t \in V(G)$, one can either conclude that G contains a minimal st-cut of size at least k, or compute a tree decomposition of a reduced graph G' of width $\mathcal{O}(k)$ such that G has a minimal st-cut of size at least k if and only if G' does. Their approach begins by constructing the block-cut tree of G in linear time and identifying the blocks B_s and B_t containing s and t, respectively. All blocks not lying on the unique path between B_s and B_t can be safely removed, yielding an equivalent instance. Moreover, they show that every block on this path either contains a $K_{2,2k}$ minor or has treewidth $\mathcal{O}(k)$. In the former case, one can guarantee the existence of a minimal st-cut of size at least k; in the latter, the bounded treewidth enables the use of dynamic programming.

In our generalization, we follow a similar line of reasoning. First, we observe that if a block B does not lie on any unique path between blocks B_{t_i} and B_{t_j} for any distinct $i, j \in [p]$, then it can be removed without affecting the minimal T-multiway cut structure. Furthermore, if any block on a path between two such terminal blocks contains a $K_{2,2k}$ minor, then by the result of Duarte et al., the graph admits a minimal t_it_j-cut of size at least k, which can be extended to a minimal T-multiway cut. Otherwise, each such block has treewidth $\mathcal{O}(k)$, and the resulting graph G' consisting only of relevant blocks has bounded treewidth.

Finally, we design a dynamic programming algorithm over a tree decomposition of G', which solves the problem in FPT time.

MaxMin Multiway Cut-Uncut when $|T_i| \leq 2$ for all $i \in [p]$, on (q,k)-unbreakable graphs.

To prove Theorem 79, we develop a branching algorithm. The key observation behind the algorithm is that if there exist sets $A_1, A_2, \ldots, A_p$ such that $a_i \in A_i$ for all $i \in [p]$, $A_i \cap A_j = \emptyset$ and $E(A_i, A_j) = \emptyset$ for $i \neq j$, each $G[A_i]$ is connected, and $b_i \notin A_j$ when $1 \leq i \neq j \leq p$, then we can consider the graph G' obtained by contracting each A_i into a single vertex a'_i. That is, we contract all edges with both endpoints in A_i. We define $T'_i = \{a'_i, b'_i\}$ where $b'_i = b_i$ if $b_i \notin A_i$, and $b'_i = a'_i$ otherwise.

For every minimal $(T'_1, \ldots, T'_p)$-multiway cut-uncut, there exists a $\mathcal{T}$-multiway cut-uncut of the same or larger size in G, which can be obtained in polynomial time. Since we are working on (q, k)-unbreakable graphs, if at least two of the sets $A_1, \ldots, A_p$ have size at least q, then every minimal $\mathcal{T}$-multiway cut-uncut has size at least k. Thus, the goal of the algorithm is to construct such sets.

The algorithm starts with $A_i = \{a_i\}$ for all $i \in [p]$. We first check whether a feasible solution exists. If not, we return a no-instance. If a feasible solution exists, we can greedily construct a minimal $\mathcal{T}$-multiway cut-uncut contained within it. If this minimal $\mathcal{T}$-multiway cut-uncut has size at least k, we return a yes-instance. Therefore, we assume it has size less than k. Deleting such a set yields exactly p connected components, each containing terminals from exactly one T_i. Since the graph is (q, k)-unbreakable, at most one of these components can have size more than q. We now focus on those T_i's that lie in connected components of size less than q.

Suppose $b_i \notin A_i$. Let Z be a solution. Then in $G - Z$, there must be a path between a_i and b_i, which implies that some edge between A_i and $N(A_i)$ is not in Z. Otherwise, such a path cannot exist. Hence, we branch on which neighboring vertex of A_i lies in the same connected component as T_i in $G - Z$. Since the component has size less than q, we have $|N(A_i)| \leq q + k$, and thus make at most $q + k$ guesses, increasing the size of some A_i by one.

If this rule is not applicable, then either $b_i \in A_i$ for all $i \in [p]$, or there exists a unique $i \in [p]$ such that $b_i \notin A_i$, with $\left|\bigcup_{j \in [p] \setminus \{i\}} N(A_j)\right| \leq q + k$ and $|A_j| < q$ for all $j \neq i$. In the former case, the instance reduces to a standard MaxMin Multiway Cut on the contracted graph G'. We separately construct an FPT algorithm for MaxMin Multiway Cut on general graphs. As MaxMin Multiway Cut is FPT, we solve this new instance on graph G' in FPT time as well.

In the second case, we proceed as follows. Since $b_i \notin A_i$ for a unique i, the set $\bigcup_{j \neq i} \delta(A_j)$ is a (not necessarily minimal) $\mathcal{T}$-multiway cut-uncut. One can greedily compute a minimal $\mathcal{T}$-multiway cut-uncut contained in this set in polynomial time. If this cut has size more than k, we return a yes-instance. Otherwise, there must be an edge in this set that is not in Z, as Z is a minimal $\mathcal{T}$-multiway cut-uncut of size at least k. We branch on which such edge is excluded from the solution and grow the corresponding A_j by including the other endpoint of that edge. In each branching step, the size of some A_i increases. Moreover, once $|A_i| = q$ for some i, the branching rules does not allow to increase its size any further. Hence, we can

bound the depth of the search tree by $(q+1)p$.

MaxMin Subset FVS **on** (q,k)**-unbreakable graphs.**

As discussed earlier this algorithm is inspired from the algorithm for MaxMin Odd Cycle Transversal on (q,k)-unbreakable graphs. In this algorithm, we provide two sufficient conditions for the existence of yes instance. The first sufficient condition is that given positive integers q,k if G is $(q,2k)$-unbreakable and there exists an induced cycle in G of length at least $2q+2$ intersecting T then G has a minimal T-subset feedback vertex set of size at least k. The proof of this condition relies on the branching algorithm for the MaxMin st-Separator problem on (q,k)-unbreakable graphs. The second condition is that if there exists a family $\mathcal{F}$ of distinct induced cycles intersecting T of G of length at most d and $|\mathcal{F}| > d(d!)(k-1)^d$ then G has a minimal T-subset feedback vertex set of size at least k.

We also make an important observation. If there exists a vertex $v \in V(G)$ that does not participate in any induced cycle of G intersecting T then v is an irrelevant vertex. Also removing such a vertex from graph returns an equivalent instance.

Note that although the two above conditions are sufficient, we cannot use them as stopping criteria as verifying this sufficient conditions cannot be done efficiently. Although this is the case, we provide a branching algorithm that given a graph on large number of vertices, either outputs one of these conditions, in which case we can return a yes instance or returns an irrelevant vertex. In the latter case, we can reduce the size of the graph by removing the found vertex. Because of deleting such vertices, the new graph may no longer be unbreakable, in which case we start solving the problem on the reduced graph completely from scratch. This result finally allows us to bound the number of vertices in the graph by $(qk)^{\mathcal{O}(q)}$. We can then solve the problem using a brute-force algorithm on this graph.

MaxMin List Allocation **parameterized by** $r+\alpha_{\max}$**.**

Surprisingly, it is much easier to construct a randomized FPT algorithm for MaxMin List Allocation when parameterized by $r+\alpha_{\max}$ compared to the above problems. Note that as r and $\alpha_{\max}$ are both bounded, so we can find a certificate of bounded size which is a subgraph of G containing at most $\binom{r}{2}\alpha_{\max}$ whose vertices can be partitioned into r parts such that there are at least α_{ij} between part V_i and V_j. Therefore, such a set can be easily

detected by color coding the vertices with r distinct colors. The rest of the vertices (not in a certificate) then can be partitioned arbitrarily.

6.2 MaxMin Multiway Cut-Uncut

In this section, we study the MaxMin Multiway Cut-Uncut problem.

Definition 30. *A feasible solution to the* MaxMin Multiway Cut-Uncut *problem is a minimal set $Z \subseteq E(G)$ such that in $G - Z$, each terminal set T_i is entirely contained within a single connected component, and for all $i \neq j$, the connected components containing T_i and T_j are distinct.*

In the Vertex-Disjoint Paths, given an undirected graph G, and a collection of terminal pairs $T_i = \{a_i, b_i\}$ for $i \in [p]$, the goal is to determine whether there exist p pairwise vertex-disjoint paths $P_1, P_2, \ldots, P_p$ in G such that each path P_i connects s_i to t_i for all $i \in [p]$.

Theorem 81. *Let $G(V, E)$ be a graph and $T_i = \{a_i, b_i\} \subseteq V(G)$ for $i \in [p]$. Then $I = (G, (T_1, T_2, \ldots, T_p))$ is a yes-instance of the* Vertex-Disjoint Paths *if and only if there is a feasible solution to* MaxMin Multiway Cut-Uncut.

Proof. In the forward direction, let us assume that I is a yes-instance of the Vertex-Disjoint Paths. It means G has pairwise vertex-disjoint paths $P_1, P_2, \ldots, P_p$ such that $T_i \subseteq V(P_i)$ for $i \in [p]$. It is clear that $E(G) \setminus \bigcup_{1 \leq i \leq p} E(P_i)$ is a $\mathcal{T}$-multiway cut-uncut.

In the backward direction, let us assume that there exists a $\mathcal{T}$-multiway cut-uncut F. We note that the condition that F is a $\mathcal{T}$-multiway cut-uncut is a guarantee of a partition of $V(G)$ into subset $S_1, S_2, \ldots, S_\ell$ such that $T_i \subseteq S_j$ for some $j \in [\ell]$ and $G[S_j]$'s are connected for $j \in [\ell]$. Since $G[S_j]$ is connected, there exists a path P^i contained in S_j such that $T_i \subseteq V(P^i)$. It is also clear that P^i's are pairwise vertex-disjoint because $\{S_1, S_2, \ldots, S_\ell\}$ is a partition of $V(G)$. Hence I is a yes-instance of the Vertex-Disjoint Paths. □

Theorem 82 ([105])**.** Vertex-Disjoint Paths *is FPT parameterized by p.*

Note that in Theorem 81, $a_i = b_i$ is allowed. Hence for each $i \in [p]$, we have $|T_i| \leq 2$. The combination of NP-completeness [83] of the Vertex-Disjoint Paths and Theorem 81

implies that determining if there is a feasible solution to MaxMin Multiway Cut-Uncut is NP-hard even when $|T_i| \leq 2$. It is easy to see that feasible solution of MaxMin Multiway Cut-Uncut exists if and only if $(G, (T_1, T_2, \ldots, T_p), k = 1)$ is a yes instance of MaxMin Multiway Cut-Uncut. Hence, MaxMin Multiway Cut-Uncut is NP-hard for fix k. Therefore we get the following corollary.

Corollary 6.2.1. MaxMin Multiway Cut-Uncut *is para-NP-hard parameterized by solution size even when* $|T_i| \leq 2$.

In the 2-Disjoint Connected Subgraphs, given a graph and two disjoint terminal sets T_1, T_2, the goal is to find disjoint connected sets S_1, S_2 such that $T_1 \subseteq S_1$ and $T_2 \subseteq S_2$. By adapting the proof technique of Theorem 81, one can easily show that $(G, (T_1, T_2))$ is a yes-instance of the 2-Disjoint Connected Subgraphs if and only if there is a feasible solution to MaxMin Multiway Cut-Uncut. It is known that 2-Disjoint Connected Subgraphs is NP-hard [110] even if $|T_1| = 2$. Hence determining if there is a feasible solution to MaxMin Multiway Cut-Uncut is NP-hard even if $p = 2$. It means MaxMin Multiway Cut-Uncut is NP-hard for $p = 2$ and $k = 1$. Therefore we get the following theorem.

Theorem 83. MaxMin Multiway Cut-Uncut *is para-NP-hard parameterized by* p *and* k *combined.*

In the MaxMin Multiway Separator-Unseparator, given an undirectered graph G, $T \subseteq V(G)$, a partition $(T_1, T_2, \ldots, T_p)$ of T and a positive integer k, the goal is to determine if there exist $X \subseteq V(G)$ of size at least k, such that X is a minimal set with the property that in $G - X$, for each $i \in [p]$, T_i is contained in a connected component, and for each $i, j \in [p]$, $i \neq j$, the connected component containing T_i is distinct from that containing T_j. We established the para-NP-hardness of MaxMin Multiway Cut-Uncut using the notion of feasible solutions of MaxMin Multiway Cut-Uncut. A similar technique can be used to derive hardness results for MaxMin Multiway Separator-Unseparator. We observe that $(G, (T_1, T_2))$ is a yes-instance of the 2-Induced Disjoint Paths if and only if there is a feasible solution to MaxMin Multiway Separator-Unseparator. It is known that 2-Induced Disjoint Paths is NP-hard [16, 53]. Consequently, deciding if there is a feasible solution to MaxMin Multiway Separator-Unseparator is NP-hard even if $p = 2$, $k = 1$ and $|T_1| = |T_2| = 2$. This leads to the following theorem.

Theorem 84. MaxMin Multiway Separator-Unseparator *is para-NP-hard parameterized by p and k combined even if $|T_1| = |T_2| = 2$.*

6.2.1 FPT algorithm for MaxMin Multiway Cut-Uncut parameterized by $k + p$ when $|T_i| \leq 2$

In this section, we show that MaxMin Multiway Cut-Uncut admits an FPT algorithm parameterized by the solution size and p combined, in the special case where each terminal set satisfies $|T_i| \leq 2$.

Theorem 76. MaxMin Multiway Cut-Uncut *is FPT parameterized by $p + k$ when $|T_i| \leq 2$ $\forall i \in [p]$.*

Let I be an instance of MaxMin Multiway Cut-Uncut where $|T_i| \leq 2$ for all $i \in [p]$. We will assume that $T_i = \{a_i, b_i\}$ where the vertices a_i and b_i may not be distinct. The goal is to reduce the task to designing an algorithm for (q, k)-unbreakable graphs. For this we first show that the problem can be expressed in CMSO (in fact in MSO).

Lemma 85. MaxMin Multiway Cut-Uncut *is CMSO-definable with a formula of length $\mathcal{O}(p^2 k)$.*

Proof. The instance (G, T, k) is a yes instance of MaxMin Multiway Cut-Uncut if and only if there exists $Z \subseteq E(G)$ of size at least k such that $G - Z$ has at least p connected components $U_1, U_2, \ldots, U_h$ where $h \geq p$ such that $T_i \subseteq V(U_i)$, and there are at least k distinct edges $e_1, \ldots, e_k$ in Z such that for each $l \in [k]$, the two endpoints of e_l are in two distinct connected components U_i and U_j for some $1 \leq i < j \leq p$. These properties of Z can be incorporated as a CMSO formula ψ as follows, where $\mathbf{conn}(U)$ is a CMSO sentence that checks whether a vertex set U induces a connected graph. The CMSO description of $\mathbf{conn}(U)$ can be, for example, found in [40].

$$\psi = \exists Z \subseteq E(G)\Bigg(\exists e_1, e_2, \ldots, e_k \in Z \Bigg(\bigwedge_{1 \leq i < j \leq k} e_i \neq e_j \Bigg)$$
$$\wedge \bigwedge_{i=1}^{p} \exists U_i \subseteq V(G)\,(a_i \in U_i \wedge b_i \in U_i \wedge \mathbf{conn}(U_i))$$
$$\bigwedge_{1 \leq i < j \leq p} \Big(\neg \mathbf{sameCC}_{G-Z}(a_i, a_j) \wedge \neg \mathbf{sameCC}_{G-Z}(a_i, b_j)$$
$$\wedge \neg \mathbf{sameCC}_{G-Z}(b_i, a_j) \wedge \neg \mathbf{sameCC}_{G-Z}(b_i, b_j) \Big)$$
$$\wedge \bigwedge_{\ell=1}^{k} \exists u, v \in V(G) \Big(\mathrm{inc}(e_\ell, u) \wedge \mathrm{inc}(e_\ell, v) \wedge u \neq v$$
$$\wedge \bigvee_{1 \leq i < j \leq p} ((u \in U_i \wedge v \in U_j) \vee (u \in U_j \wedge v \in U_i)) \Big) \Bigg)$$

Note that we define $\mathbf{same\ CC}_{G-Z}(u, v) = \exists U \subseteq V(G) \Big((u \in U) \wedge (v \in U) \wedge \mathbf{conn}_{G-Z}(U) \Big)$. □

Theorem 79. *For positive integers $q, k \geq 1$,* MaxMin Multiway Cut-Uncut *when $|T_i| \leq 2$ for all $i \in [p]$, on (q,k)-unbreakable graphs on n vertices can be solved in time $f(q,k) \cdot n^{\mathcal{O}(1)}$, for some function f that depends only on q and k.*

Proof. We can assume that the input graph G is connected. Note that our goal is to decide if there exists a minimal $\mathcal{T}$-multiway cut-uncut of size at least k. We begin our algorithm by checking if there is a feasible solution for the input instance. Due to Theorem 82, this can be done in FPT time. If there is no feasible solution then we can return no instance for our problem as well. Let us assume that we obtain a feasible solution Z'. Then we can find a minimal $\mathcal{T}$-multiway cut-uncut Z that is contained in Z' in polynomial time. If we get $|Z| \geq k$ then we can return a yes instance. In the other case, we can assume that $|Z| < k$. If Z is a minimal $\mathcal{T}$-multiway cut-uncut of size less than k then we know that $G - Z$ contains at most k connected components. In turn, this implies that $p \leq k$. Since Z is minimal, we must obtain exactly p connected components in $G - Z$ where each connected component contains vertices from exactly one T_i for each $i \in [p]$. As the input graph G is (q,k)-unbreakable,

there is exactly one connected components in $G - Z$ whose size is at least q. We further develop notations that will be helpful in understanding the final branching algorithm.

We design an algorithm that maintains a tuple $(G, (A_1, b_1), (A_2, b_2), \ldots, (A_p, b_p), q, k)$ based on the branching technique where G is a (q, k)-unbreakable graph, q, k are positive integers and for each $i \in [p], A_i \subseteq V(G)$. Additionally, A_i's satisfy the following properties.

1. For each $i \in [p]$, $a_i \in A_i$,
2. $A_i \cap A_j = \emptyset$ for $i \neq j$,
3. $E(A_i, A_j) = \emptyset$ for $i \neq j$
4. $G[A_i]$ is connected for each $i \in [p]$ and
5. $b_i \notin A_j$ when $1 \leq i, j \leq p$ and $i \neq j$.

Let G' be the graph obtained by contracting each set A_i into a vertex a'_i, i.e., by contracting all edges with both endpoints in A_i. Define $T'_i = \{a'_i, b'_i\}$, where $b'_i = b_i$ if $b_i \notin A_i$, and $b'_i = a'_i$ otherwise. The weight of T'_i is defined as $\mathtt{wt}(T'_i) = |A_i|$ if $b_i \in A_i$, and $\mathtt{wt}(T'_i) = |A_i|+1$ if $b_i \notin A_i$.

Given an instance I where the A_i's satisfies conditions 1-5 and also admits a feasible solution for the partition $\mathcal{T}' = (T'_1, T'_2, \ldots, T'_p)$ in G' is called a valid instance. We design a branching algorithm that outputs a minimal $\mathcal{T}$-multiway cut-uncut in G, which is disjoint from $\bigcup_{i=1}^{p} E(G[A_i])$, and has size at least k, if it exists. For a valid instance I, we define its measure $\mu(I) = q - \mathrm{Max}_2\left(\mathtt{wt}(T'_1), \ldots, \mathtt{wt}(T'_p)\right)$ where $\mathrm{Max}_2(\cdot)$ denotes the second largest value in the given set of values.

Lemma 86. *Let $(G, (A_1, b_1), (A_2, b_2), \ldots, (A_p, b_p), q, k)$ be a valid instance. If $I_1 = (G', (T'_1, T'_2, \ldots, T'_p), k)$ is a yes instance of* MaxMin Multiway Cut-Uncut *in G' then $I_2 = (G, (T_1, T_2, \ldots, T_p), k)$ is a yes instance of* MaxMin Multiway Cut-Uncut.

Proof. Recall that G' be the graph obtained from G by contracting each set A_i into a single vertex a'_i, that is, by contracting all edges with both endpoints in A_i. Define a vertex map

$\phi : V(G) \to V(G')$ as follows:

$$\phi(v) = \begin{cases} a'_i & \text{if } v \in A_i \text{ for some } i \in \{1, \ldots, p\}, \\ v & \text{otherwise.} \end{cases}$$

Now, define a map $f : E(G) \setminus \bigcup_{i=1}^{p} E(G[A_i]) \to E(G')$, which maps each edge $e = uv$ to the edge $f(e) = \phi(u)\phi(v) \in E(G')$. In other words, the edge e in G, with endpoints u and v, is mapped to the edge between the images of u and v under the contraction operation, with each vertex $v \in A_i$ replaced by the contracted vertex a'_i and vertices outside all A_i remaining unchanged. Note that f may be a many-to-one function; that is, multiple edges in $E(G) \setminus \bigcup_{i=1}^{p} E(G[A_i])$ may be mapped to the same edge in $E(G')$ if they are incident to vertices that are merged during the contraction. For an edge $e' \in E(G')$, the preimage under f is given by

$$f^{-1}(e') = \left\{ e \in E(G) \setminus \bigcup_{i=1}^{p} E(G[A_i]) \text{ such that } f(e) = e' \right\},$$

which represents the set of edges in G that are contracted or merged into the edge $e' \in E(G')$ in the contracted graph.

Let $I_1 = (G', (T'_1, T'_2, \ldots, T'_p), k)$ be a yes instance of MaxMin Multiway Cut-Uncut. This means that we have a minimal $\mathcal{T}'$-multiway cut-uncut in G', say Z_2, of size at least k. We note that $Z_1 := \{f^{-1}(e') \mid e' \in Z_2\}$ is a minimal $\mathcal{T}$-multiway cut-uncut in G. Since $|Z_2| \geq k$, the minimal $\mathcal{T}$-multiway cut-uncut obtained from the preimage of Z_2 must have size at least k. Therefore, $I_2 = (G, (T_1, T_2, \ldots, T_p), k)$ is a yes instance of MaxMin Multiway Cut-Uncut. □

Lemma 87. *Let $I = (G, (A_1, b_1), (A_2, b_2), \ldots, (A_p, b_p), q, k)$ be a valid instance. If $\mu(I) \leq 0$, then every $\mathcal{T}$-multiway cut-uncut, disjoint from $\bigcup_{i=1}^{p} E(G[A_i])$, has size at least k.*

Proof. If $\mu(I) \leq 0$, then $q \leq \mathrm{Max}_2\left(\mathtt{wt}(T'_1), \mathtt{wt}(T'_2), \ldots, \mathtt{wt}(T'_p)\right)$. For the sake of contradiction, let us assume that there exists a $\mathcal{T}$-multiway cut-uncut Z in G, which is disjoint from $\bigcup_{i=1}^{p} E(G[A_i])$, and has size strictly less than k. Note that $G \setminus Z$ contains at least two distinct connected components containing $A_i \cup \{b_i\}$ and $A_j \cup \{b_j\}$ of size at least q. This is a witnessing separation that G is (q, k)-breakable, which is a contradiction. □

The safeness of Reduction Rule 4 is immediate from Lemma 87.

Reduction Rule 4. *If $\mu(I) \leq 0$, then report a yes instance.*

The Algorithm initializes $A_i = \{a_i\}$ for each $i \in [p]$. To solve an instance $I = (G, (A_1, b_1), (A_2, b_2), \ldots, (A_p, b_p), q, k)$, we provide the following branching rules.

Branching Rule 1. *Let $I = (G, (A_1, b_1), \ldots, (A_p, b_p), q, k)$ be a valid instance. If there exists $i \in [p]$ such that $b_i \notin A_i$, $|A_i| < q$, and $|N(A_i)| \leq q + k$, then for each $x \in N(A_i)$, we recursively solve the instance $(G, (A_1, b_1), \ldots, (A_i \cup \{x\}, b_i), \ldots, (A_p, b_p), q, k)$, provided it is valid.*

Lemma 88. *The Branching Rule 1 is correct.*

Proof. Let us assume that Z is a minimal $\mathcal{T}$-multiway cut-uncut of size at least k such that Z is disjoint from $\bigcup_{i=1}^{p} E(G[A_i])$. Note that $G \setminus Z$ contains a connected component U_i that contains the vertices $A_i \cup \{b_i\}$ such that $b_i \notin A_i$. Therefore, there must exist an edge in $\delta(A_i)$ which is not contained in Z. Otherwise, a_i will not be reachable to b_i. Let the endpoint of that edge not in A_i is x. As the edge is not contained in Z, we must have $x \in U_i$ in $G \setminus Z$. Therefore, Z is also a minimal $\mathcal{T}$-multiway cut-uncut of size at least k such that Z is disjoint from $\bigcup\limits_{j=1,\ j\neq i}^{p} \Big(E(G[A_j]) \cup E(G[A_i \cup \{x\}]) \Big)$.

In the backward direction, let $(G, (A_1, b_1), (A_2, b_2), \ldots, (A_i \cup \{x\}, b_i), \ldots, (A_p, b_p), q, k)$ reports a solution Z. Since $A_1, A_2, \ldots, A_i \cup \{x\}, \ldots, A_p$ satisfies the desired properties of a valid instance, Z is also a minimal $\mathcal{T}$-multiway cut-uncut. □

Lemma 89. *If Reduction Rule 4 and Branching Rule 1 are not applicable, then one of the following holds:*

(1) *$b_i \in A_i$ for all $i \in [p]$; or*

(2) *There exists a unique $i \in [p]$ such that $b_i \notin A_i$, and moreover, $\left|\bigcup_{j \in [p] \setminus \{i\}} N(A_j)\right| \leq q + k$ and $|A_j| < q$ for all $j \in [p] \setminus \{i\}$.*

Proof. We begin by finding a minimal $\mathcal{T}'$-multiway cut-uncut in G' say Z'. We note that $Z'' := \{f^{-1}(e') \mid e' \in Z'\}$ is a minimal $\mathcal{T}$-multiway cut-uncut in G, which is disjoint from $\bigcup_{j=1}^{p} E(G[A_j])$. If $|Z''| \geq k$, then we are done. So we assume $|Z''| < k$. Since Z'' is a

minimal $\mathcal{T}$-multiway cut-uncut, removing Z'' from G separates each T_j into its own connected component. For each $j \in [p]$, let C_j be the connected component containing T_j in $G - Z''$. Note that no two of the components C_j can have size at least q, due to the (q,k)-unbreakability of the graph G. We also note that $A_j \subseteq V(C_j)$ for each $j \in [p]$. Therefore, we may assume that exactly one of the components has size at least q. Without loss of generality, assume that $|V(C_\ell)| \geq q$ for some $\ell \in [p]$. We also note that the (q,k)-unbreakability of the graph G makes sure that $\sum_{j=1,\ j\neq\ell}^{p} |V(C_j)| < q$. For each $j \in [p]$, we know $A_j \subseteq V(C_j)$, hence by the condition that $\sum_{j=1\ j\neq\ell}^{p} |V(C_j)| < q$, we conclude $|A_j| < q$ for $j \in [p] \setminus \ell$. From the conditions $\sum_{j=1\ j\neq\ell}^{p} |V(C_j)| < q$ and $|Z''| \leq k$, we have $|N(C_j)| < q + k$ for all $j \in [p] \setminus \{\ell\}$. Hence we have $|N(A_j)| \leq |N(C_j)| < q + k$ for all $j \in [p] \setminus \{\ell\}$. For all $j \in [p] \setminus \{\ell\}$, $b_j \in A_j$, otherwise the branching rule 1 could have been applied. Now, if $b_\ell \in A_\ell$, then the first condition is true; otherwise, the second condition is true. □

Lemma 90. *If $b_i \in A_i$ for all $i \in [p]$ then one can use the FPT algorithm for* Maximum Minimal Multiway Cut *to solve this instance.*

Proof. Note that if $b_i \in A_i$ for all $i \in [p]$ then determining if there is a $\mathcal{T}$-multiway cut-uncut of size at least k reduces to the MaxMin Multiway cut problem as $|T'_i| = 1$ for all $i \in [p]$. One can use the FPT algorithm in Subsection 6.3.2 to solve the problem in FPT time on the graph G'. □

Branching Rule 2. *Given an instance $I = (G, (A_1, b_1), (A_2, b_2), \ldots, (A_p, b_p), q, k)$, if there exists a unique $i \in [p]$ such that $b_i \notin A_i$ then for each vertex $x \in N(A_\ell)$ with $\ell \neq i$, we recursively solve the instance $(G, (A_1, b_1), (A_2, b_2), \ldots, (A_\ell \cup \{x\}, b_\ell), \ldots, (A_p, b_p), q, k)$ if it is a valid instance.*

Lemma 91. *The Branching Rule 2 is correct.*

Proof. Let us assume that I is a yes instance, that is, there exists a minimal $\mathcal{T}$-multiway cut-uncut Z of size at least k in G, that is disjoint from $\bigcup_{i=1}^{p} E(G[A_i])$. We note that $\bigcup_{j=1,\ j\neq i}^{p} \delta(A_j)$ is a $\mathcal{T}$-multiway cut-uncut. We can construct a minimal $\mathcal{T}$-multiway cut-uncut contained in this set in polynomial time. Lets call it Z'. If $|Z'| \geq k$ then we can return a yes instance. If not then there exists an edge e in Z' that is not in Z. Let the edge e has an endpoint

outside $\bigcup_{j=1,\ j\neq i}^{p} A_j$ is denoted by x. Let $x \in N(A_\ell)$ for some $\ell \neq i$ then Z is also a minimal $\mathcal{T}$-multiway cut-uncut such that Z is disjoint from $\bigcup_{j=1,\ j\neq \ell}^{p} \Big(E(G[A_j])\Big) \cup E(G[A_\ell \cup \{x\}])$.

In the backward direction, let Z be a $\mathcal{T}$-multiway cut-uncut such that Z is disjoint from $\bigcup_{j=1,\ j\neq i}^{p} \Big(E(G[A_j]) \cup E(G[A_i \cup \{x\}])\Big)$. Then clearly Z is a minimal $\mathcal{T}$-multiway cut-uncut such that Z is disjoint from $\bigcup_{i=1}^{p} E(G[A_i])$. □

Lemma 92. *The branching algorithm runs for at most $(q+1)p$ iterations.*

Proof. Note that each time we apply a branching rule, the size $\mathtt{wt}(T'_i)$ increases by one for some $i \in [p]$. In the Branching Rule 1, $\mathtt{wt}(T'_i)$ increases only for those i for which we have $b_i \notin A_i$, $|A_i| < q$ and $|N(A_i)| \leq q + k$. Note that the conditions that $b_i \notin A_i$ and $|A_i| < q$ makes sure that $\mathtt{wt}(T'_i) \leq q$ before we apply the branching rule and $\mathtt{wt}(T'_i)$ increases by 1 in each iteration. Similarly in the Branching Rule 2, $\mathtt{wt}(T'_\ell)$ increases only for those ℓ for which we have $b_\ell \in A_\ell$, $|A_\ell| < q$. Note that the conditions that $b_\ell \in A_\ell$ and $|A_i| < q$ makes sure that $\mathtt{wt}(T'_\ell) \leq q$ before we apply the branching rule and $\mathtt{wt}(T'_\ell)$ increases by 1 in each iteration. Hence in each branching rule, for any $j \in [p]$, $\mathtt{wt}(T'_j)$ changes only if $\mathtt{wt}(T'_j) \leq q$. In other words, once the $\mathtt{wt}(T'_j)$ takes the value $q+1$, $\mathtt{wt}(T'_j)$ stops increasing. Algorithm initializes with $\mathtt{wt}(T'_j) \leq 2$ for each $j \in [p]$. After $(q+1)p$ iterations, there must exist $j \in [p]$ such that $\mathtt{wt}(T'_j) = q+1$. Without loss of generality, we assume that $\mathtt{wt}(T'_1) = q+1$. We have already noted that each iteration increases $\mathtt{wt}(T'_j)$ for atleast one j and the maximum value $\mathtt{wt}(T'_1)$ can take is $q+1$. Because of remaining $(q+1)p - (q+1)$ iterations, values of $\mathtt{wt}(T'_j)$ changes for $j \in [p] \setminus \{1\}$. Hence there must exist $j \in [p] \setminus \{1\}$ such that $\mathtt{wt}(T'_j) = q$. This implies after $(q+1)p$ iterations, $\mu(I) \leq 0$. □

6.3 MaxMin Multiway Cut

A *cut*, denoted by $[S_1, S_2, \ldots, S_t]$, is a partition of the vertex set $V(G)$ of a graph G into disjoint subsets $S_1, S_2, \ldots, S_t$ such that $V(G) = \bigcup_{i=1}^{t} S_i$. For any two parts S_i and S_j, let $\partial(S_i, S_j)$ denote the set of edges with one endpoint in S_i and the other in S_j. The set $\bigcup_{1\leq i<j\leq t} \partial(S_i, S_j)$ is called the *cut-set* associated with the cut $[S_1, S_2, \ldots, S_t]$. Given a graph

G and a terminal set $T = \{t_1, t_2, \ldots, t_p\} \subseteq V(G)$, a nonempty set of edges $F \subseteq E(G)$ is a minimal T-multiway cut if and only if there exists a cut $[S_1, S_2, \ldots, S_p]$ of G such that:

- for all $i \in [p]$, $t_i \in S_i$,
- each subgraph $G[S_i]$ is connected, and
- $F = \bigcup_{1 \leq i < j \leq p} \partial(S_i, S_j)$.

In this case, we say that $[S_1, S_2, \ldots, S_p]$ is the *cut defined by* F. To prove our theorem, we will use some lemmas given in [47]. Note that they refer to a minimal st-cut as an st-bond. For convenience, we will use the term *minimal st-cut* instead of st-bond throughout this chapter.

Lemma 93 ([47])**.** *Let G is a 2-vertex-connected graph and $s, t \in V(G)$. There is a polynomial-time algorithm that either concludes that G has minimal st-cut of size at least k, or outputs a tree decomposition of G of width $\mathcal{O}(k)$.*

Now, we give a lemma which concludes that if a graph G and $T = \{t_1, t_2, \ldots, t_p\} \subseteq V(G)$ are given, then the existence of minimal t_it_j-bond of size at least k for some $t_i \neq t_j$ guarantees the existence of minimal T-multiway cut of size at least k.

Lemma 94. *For a graph G and $T = \{t_1, t_2, \ldots, t_p\} \subseteq V(G)$, if G has a minimal t_it_j-cut of size at least k for some $t_i, t_j \in T$, then G has a minimal T-multiway cut of size at least k.*

Proof. Let G has a minimal t_it_j-cut, say F, of size at least k for some $t_i, t_j \in T$. Let $[S_i, S_j]$ be the cut defined by F. Note that $G[S_i]$ and $G[S_j]$ must be connected and $\partial(S_i) = \partial(S_j) = F$. Suppose S_i and S_j contains α and β number of terminals of T, respectively. Let the terminals in S_i be relabeled as $t_{i_1}, t_{i_2}, \ldots, t_{i_\alpha}$, and those in S_j as $t_{j_1}, t_{j_2}, \ldots, t_{j_\beta}$, respectively. Let $T_i := \{t_{i_1}, t_{i_2}, \ldots, t_{i_\alpha}\}$ and $T_j := \{t_{j_1}, t_{j_2}, \ldots, t_{j_\beta}\}$. We then compute a minimal T_i-multiway cut, say F_i in the subgraph $G[S_i]$, and similarly, a minimal T_j-multiway cut, say F_j in the subgraph $G[S_j]$, using a greedy approach.

We claim that $F \cup F_i \cup F_j$ is a minimal T-multiway cut in G. It is clear that $F \cup F_i \cup F_j$ is a T-multiway cut in G. Let $[S_{i_1}, S_{i_2}, \ldots, S_{i_\alpha}]$ and $[S_{j_1}, S_{j_2}, \ldots, S_{j_\beta}]$ be cuts defined by

F_i and F_j, respectively. Observe that each part of these cuts are connected and contains exactly one terminal from T. It is clear that $F \cup F_i \cup F_j$ is a cut-set with the cut $[S_{i_1}, S_{i_2}, \ldots, S_{i_\alpha}, S_{j_1}, S_{j_2}, \ldots, S_{j_\beta}]$. Since all components are connected and each contains a distinct terminal of T, $[S_{i_1}, S_{i_2}, \ldots, S_{i_\alpha}, S_{j_1}, S_{j_2}, \ldots, S_{j_\beta}]$ is cut defined by $F \cup F_i \cup F_j$. Hence, $F \cup F_i \cup F_j$ is a minimal T-multiway cut of size at least k. □

We now state a theorem that generalizes a result (see Corollary 3) presented in [47]. The proof technique we use is also closely aligned with the one given there.

Theorem 95. *Given a graph G, $T = \{t_1, t_2, \ldots, t_p\} \subseteq V(G)$, and an integer k, there exists a polynomial-time algorithm that either concludes that G has a minimal T-multiway cut of size at least k or outputs a subgraph G' of G together with a tree decomposition of G' of width $\mathcal{O}(k)$, such that G' has a minimal T-multiway cut of size at least k if and only if G has a minimal T-multiway cut of size at least k.*

Proof. We begin by computing the block-cut tree of G in linear time [36]. Let B_i denote the block of G that contains the terminal t_i. We remove each block that does not lie on any path between a pair (B_i, B_j) for $i \neq j$ in the block-cut tree of G. Let G' be the resulting subgraph of G.

Note that every block of G' must lie on the path between some pair (B_i, B_j) for $i \neq j$. Let B be a block that lies on the path from B_i to B_j in the block-cut tree. Consider the vertices t'_i and t'_j in B that are closest to t_i and t_j, respectively. Using Lemma 93, we can, in polynomial time, either conclude that B has a minimal $t'_i t'_j$-cut of size at least k, or compute a tree decomposition of B of width at most $\mathcal{O}(k)$. In the former case, since G contains a $t'_i t'_j$-cut of size at least k, it also contains a $t_i t_j$-bond of size at least k, and by Lemma 94, it follows that G admits a minimal T-multiway cut of size at least k.

We now describe how to construct a tree decomposition of G'. We start by taking the union of the tree decompositions of all blocks in G'. For each cut vertex u of G', we introduce a bag $\{u\}$. Then, for every cut vertex u and each block B containing u, we connect the bag $\{u\}$ to some bag in the tree decomposition of B that contains u. This results in a tree decomposition of G' in which every bag has size at most $\mathcal{O}(k)$.

Finally, we show that G' has a minimal T-multiway cut of size at least k if and only if G has a minimal T-multiway cut of size at least k. Suppose G has a minimal T-multiway

cut, say Z of size at least k. Since G' is an induced subgraph of G, Z must be T-multiway cut in G'. Let $e \in Z$, then in $G - (Z \setminus \{e\}$, there is a $t_i t_j$-path for some $t_i, t_j \in T$. Observe that such a path must pass through the block of G that contains e, and since G' includes all blocks on paths between terminal-containing blocks B_i and B_j, this path also exists in $G' - (Z \setminus \{e\})$. Therefore, Z is also a minimal T-multiway cut in G'.

Suppose G' has a minimal T-multiway cut, say Z' of size at least k. We first show that Z' is also a T-multiway cut in G. For the sake of contradiction, let us assume that Z' is not a T-multiway cut in G, that means, in $G - Z'$, there exists a $t_i t_j$-path for some $t_i, t_j \in T$. Note that such a path must contain a sub-path that contains edges only appearing in the blocks that are in the $B_i - B_j$ path in the block-tree. Hence in $G' - Z'$, there is a $t_i t_j$-path in G', which is a contradiction. Hence Z' is a T-multiway cut in G. Since Z' is a minimal T-multiway cut in G', for each $e' \in Z'$, in $G' - (Z' \setminus \{e'\}$ there must exist a $t_i t_j$-path for some $t_i, t_j \in T$, say P'. Since G is a supergraph of G', the same path P' also exists in $G - (Z' \setminus \{e'\}$. This implies Z' is a minimal T-multiway cut in G.

□

6.3.1 MaxMin Multiway Cut parameterized by Treewidth

In the following, let T be a tree decomposition of a graph G. We refer to a node of T by t, and denote by X_t the bag associated with node t. Without loss of generality, we assume that T is an extended nice tree decomposition, which means that it has a designated root node r with $X_r = \emptyset$, and all edges in the decomposition tree are directed towards r. Moreover, each node t falls into one of the following five categories: Leaf, Introduce vertex, Introduce edge, Forget vertex, or Join. For each node t, we define G_t as the subgraph of G that includes all vertices and edges introduced at t or at a descendant of t in the tree T.

Theorem 96. *Given a nice tree decomposition of a graph G with width tw, one can find a maximum size minimal T-multiway cut in time $2^{\mathcal{O}(\mathsf{tw} \log \mathsf{tw})} \cdot \mathsf{tw} \cdot n$, where n is the number of vertices of G.*

Proof. Let F be a minimal T-multiway cut of G, and $[S_1, S_2, \ldots, S_p]$ be the cut defined by F. Set $S_i^t = S_i \cap X_t$ for $i \in [p]$. The removal of F partitions $G_t[S_i]$ into a set $C_{F,i}^t$ of connected

components. Note that $C^t_{F,i}$ defines the partition of S^t_i, denoted by ρ^t_i, where the intersection of each connected component of $C^t_{F,i}$ with S^t_i corresponds to one part of ρ^t_i.

We define a table for which an entry $c[t, U_1, U_2, \ldots, U_p, \rho_1, \rho_2, \ldots, \rho_p]$ is the size of a largest cut-set (partial solution) with p parts of the subgraph G_t. Each $U_i \subseteq X_t$ denotes the set of vertices assigned to the i^{th} part. Each ρ_i is a partition of U_i induced by the connected components in G_t after removing the partial solution. We require that for all $i \in [p]$, the terminal t_i does not appear in $\bigcup_{j\in[p],j\neq i} U_j$. If there is no such partial solution then $c[t, U_1, U_2, \ldots, U_p, \rho_1, \rho_2, \ldots, \rho_p] = -\infty$. In particular, if $t_i \in U_j$ for some $i \neq j$, then $c[t, U_1, U_2, \ldots, U_p, \rho_1, \rho_2, \ldots, \rho_p] = -\infty$.

For the case that U_i is empty, two special cases may occur: either $S_i \cap V(G_t) = \emptyset$, in which case there are no connected components in $C^t_{F,i}$, and thus $\rho_i = \emptyset$; or $C^t_{F,i}$ has only one connected component which does not intersect X_t, i.e., $\rho_i = \{\emptyset\}$, this case means that the connected component in $C^t_{F,i}$ was completely forgotten. Note that we do not need to consider the case $\{\emptyset\} \subsetneq \rho_i$ since it would imply a disconnected solution. Then the value of a maximum-size minimal T-multiway cut in the connected graph G corresponds to the root entry $c[r, \emptyset, \emptyset, \ldots, \emptyset, \{\emptyset\}, \{\emptyset\}, \ldots, \{\emptyset\}]$. To describe the dynamic programming algorithm, it suffices to present the recurrence relations corresponding to each node type in the tree decomposition.

Leaf node : In this case, $X_t = \emptyset$. Since for this case G_t is empty, there can be no connected components, so having $\rho_i = \emptyset \ \forall \ i$ is the only feasible choice.

$$c[t, U_1, U_2, \ldots, U_p, \rho_1, \rho_2, \ldots, \rho_p] = \begin{cases} 0 & \text{if } U_i = \emptyset, \rho_i = \emptyset \ \forall \ i \\ -\infty & \text{otherwise} \end{cases}$$

Note that the recurrence relation can be solved in $\mathcal{O}(p)$ time

Introduce vertex node : In this case, v is in exactly one part, say U_i. If $v = t_j$ for $j \neq i$, then $c[t, U_1, U_2, \ldots, U_p, \rho_1, \rho_2, \ldots, \rho_p] = -\infty$. Since v got introduced in this step, v is an isolated vertex. Thus, a partial solution on t induces a partial solution on t', excluding v from its part U_i.

$c\left[t, U_1, \ldots, U_i, \ldots, U_p, \rho_1, \ldots, \rho_i, \ldots, \rho_p\right] =$

$$\begin{cases} c\left[t', U_1, \ldots, U_i \backslash \{v\}, \ldots, U_p, \rho_1, \ldots, \rho_i \backslash \{\{v\}\}, \ldots, \rho_p\right] & \text{if } \{v\} \in \rho_i \\ -\infty & \text{if } \{v\} \notin \bigcup_i \rho_i \end{cases}$$

Note that the recurrence relation can be solved in $\mathcal{O}(\mathsf{tw})$ time.

Introduce edge node : In this case, either the edge $\{u, v\}$ that is being inserted is incident with one vertex of two different parts, or the two endpoints are at the same part. In the former case, a solution on t corresponds to a solution on t' with the same partitions, but with value increased. In the latter case, edge $\{u, v\}$ may connect two connected components of a partial solution on t'.

$c\left[t, U_1, U_2, \ldots, U_p, \rho_1, \rho_2, \ldots, \rho_p\right] =$

$$\begin{cases} c\left[t', U_1, U_2, \ldots, U_p, \rho_1, \rho_2, \ldots, \rho_p\right] + 1 & \text{if } u \in U_i \text{ and } v \in U_j \text{ for } i \neq j, \\ \max_{\rho_i'} \left\{\left[t', U_1, U_2, \ldots, U_p, \rho_1, \ldots, \rho_i', \ldots, \rho_p\right]\right\} & \text{if } \{u, v\} \subseteq U_i \text{ for some } i \in [p] \end{cases}$$

Here, ρ_i' spans over all refinements of ρ_i such that the union of the parts containing u and v results in the partition ρ_i.

Note that we are considering all refinements of ρ_i. Let us assume that ρ_i contains α number of elements of X_t. In the worst case (when the partition is single block of size α), the number of refinements is B_α. Since $B_\alpha = 2^{\mathcal{O}(\alpha \log \alpha)}$ and $\alpha \leq \mathsf{tw} + 1$, the number of refinements is $2^{\mathcal{O}(\mathsf{tw} \log \mathsf{tw})}$. Hence the recurrence relation can be solved in $2^{\mathcal{O}(\mathsf{tw} \log \mathsf{tw})}$ time.

Forget node : In this case, the forgotten vertex v is in exactly one (say i^{th}) part of the cut for a partial solution induced on t. Thus, v must be in the connected component which contains some part of ρ_i. We select the possibility that maximizes the value

$c\left[t, U_1, U_2, \ldots, U_p, \rho_1, \rho_2, \ldots, \rho_p\right] =$

$$\max_{1 \leq i \leq p, \rho_i'} c\left[t', U_1, \ldots, U_i \cup \{v\}, \ldots, U_p, \rho_1, \ldots, \rho_i', \ldots, \rho_p\right]$$

Here, ρ_i' spans over all partitions obtained from ρ_i by adding v in some part of ρ_i (if $\rho_i = \{\emptyset\}$ then $\rho_i' = \{v\}$). Note that the recurrence relation can be solved in $\mathcal{O}(\mathsf{tw})$ time.

Join node: This node represents the join of two subgraphs $G_{t'}$ and $G_{t''}$ and $X_t = X_{t'} = X_{t''}$. By counting the cut-set edges contained in $G_{t'}$ and in $G_{t''}$, each edge is counted at least once, but edges in X_t are counted twice. Thus

$$c\left[t, U_1, U_2, \ldots, U_p, \rho_1, \rho_2, \ldots, \rho_p\right] = \max_{\rho_i', \rho_i'' \mid\ i \in [p]} \Big\{ c\left[t', U_1, U_2, \ldots, U_p, \rho_1', \rho_2', \ldots, \rho_p'\right] + c\left[t'', U_1, U_2, \ldots, U_p, \rho_1'', \rho_2'', \ldots, \rho_p''\right] - |\{\{u,v\} \in E \mid u \in U_i, v \in U_j \text{ for } i \neq j\}| \Big\}$$

We say ρ_i' and ρ_i'' are compatible given ρ if transitive closure of the union of ρ_i' and ρ_i'' is equal to ρ_i. In the join node, for $i \in \{1, 2, \ldots p\}$, we consider combinations of ρ_i' with ρ_i'' such that ρ_i' and ρ_i'' are compatible given ρ_i. If $\rho_i = \{\emptyset\}$ then either $\rho_i' = \{\emptyset\}$ and $\rho_i'' = \emptyset$; or $\rho_i' = \emptyset$ and $\rho_i'' = \{\emptyset\}$. Also, if $\rho_i = \emptyset$ then $\rho_i' = \emptyset$ and $\rho_i'' = \emptyset$. Note that to solve the recurrence relation at join node, given any tuple $c\left[t, U_1, U_2, \ldots, U_p, \rho_1, \rho_2, \ldots, \rho_p\right]$, we need to consider the tuples in the children node such that ρ_i' and ρ_i'' must form a finer partition of ρ_i for all $i \in [p]$ and the transitive closure of the union of ρ_i' and ρ_i'' is equal to ρ_i. Therefore, it would take at most $2^{\mathcal{O}(\mathsf{tw} \log \mathsf{tw})}.n^{\mathcal{O}(1)}$ time to get the value corresponding to any tuple.

The number of nodes in the tree decomposition is $\mathcal{O}(\mathsf{tw} \cdot n)$. Note that we are partitioning X_t into p parts. Then, for each i, rho_i is a partition of U_i. Basically, each ρ_i is refinements of U_i. In the worst case, we are considering all partitions of X_t. Since $|X_t| \leq \mathsf{tw} + 1$. The number of partitions is at most $B_{\mathsf{tw}+1} \leq 2^{\mathcal{O}(\mathsf{tw} \log \mathsf{tw})}$. Hence, the total size of the dynamic programming table is at most $2^{\mathcal{O}(\mathsf{tw} \log \mathsf{tw})} \cdot \mathsf{tw} \cdot n$. The correctness of the recurrence relations follows directly from the definitions and case analysis for each node type.

□

6.3.2 MaxMin Multiway Cut parameterized by Solution size

By using Theorem 95, we can, in polynomial-time either conclude that G has a minimal T-multiway cut of size at least k or get a subgraph G' of G together with a tree decomposition of G' of width $\mathcal{O}(k)$, such that G' has a minimal T-multiway cut of size at least k if and only if G has a minimal T-multiway cut of size at least k. In the former case, we can return a yes-instance. In the latter case we proceed by applying Theorem 96, which allows us to find the maximum size minimal T-multiway cut in G' in time $2^{\mathcal{O}(\mathsf{tw}\log\mathsf{tw})} \cdot \mathsf{tw} \cdot n$. Note that this gives an upper bound on running time of our FPT as $2^{\mathcal{O}(k\log k)} \cdot k \cdot n$. If maximum size minimal T-multiway cut in G' is greater than or equal to k, then return yes instance, otherwise return no instance.

6.4 MaxMin Subset Feedback Vertex Set

In this section, we prove Theorem 77. To prove Theorem 77, we first show that the problem is CMSO definable (Lemma 97). Using Proposition 74, one can reduce to solving this problem on $(q, 2k)$-unbreakable graphs. On $(q, 2k)$-unbreakable graphs, we then list and prove two sufficient conditions (Lemmas 100 and 101) which always imply a yes instance (in fact the second one implies a yes instance even when the input graph is not $(q, 2k)$-unbreakable). We also make an observation about irrelevant vertices that can be deleted without changing the solution of the instance (Observation 6.4.1). Even though checking whether any one of these sufficient conditions hold or finding these irrelevant vertices, may not be efficient, nonetheless we design an FPT algorithm that correctly concludes that at least one of the sufficient conditions is met, or outputs the set of all the irrelevant vertices, whenever the number of vertices in the graph is strictly more than a number with is a function of q and k (Theorem 99). In the case when the irrelevant vertices are outputted, deleting them reduces the size of the graph but the resulting graph may not be $(q, 2k)$-unbreakable. In this case, we start from the beginning and solve the problem from scratch (on general graphs). If none of the above hold, then the number of vertices in the graph is bounded, and we can solve the problem using brute-force.

Lemma 97. MaxMin Subset Feedback Vertex Set *is CMSO-definable by a formula of length* $\mathcal{O}(k^2)$.

Proof. Let (G, T, k) be an instance of the problem. We seek a set $Z \subseteq V(G)$ of size at least k such that no cycle in $G - Z$ contains a vertex from T, but for each $v \in Z$, removing vertex v from Z results in the existence of a cycle containing a vertex from T.

Alternately, suppose there exists a set $Z \subseteq V(G)$ that may be arbitrarily large. This set Z must satisfy the following conditions. First, it contains at least k distinct vertices, which we denote as $v_1, v_2, \ldots, v_k$. Second, no cycle in the graph $G - Z$ includes any vertex from T. Finally, for each $i \in [k]$, if we remove v_i from Z, the resulting graph $G - (Z \setminus \{v_i\})$ must contain a cycle that includes at least one vertex from T. Then Z may not be a minimal T-subset feedback vertex set but it always contains a minimal T-subset feedback vertex of size at least k. In fact, (G, T, k) is a yes instance if and only if such a set Z exists. These properties of Z can be incorporated as a CMSO formula ψ as follows.

We now translate these conditions into a CMSO formula.

$$\begin{aligned}\varphi \equiv \exists Z \subseteq V(G) \Bigg(& \exists v_1, v_2, \ldots, v_k \in Z \left(\bigwedge_{1 \leq i < j \leq k} v_i \neq v_j \right) \\ & \wedge \forall C \subseteq V(G) \Big(\mathbf{IsCycle}(C, G - Z) \rightarrow C \cap T = \emptyset \Big) \\ & \wedge \bigwedge_{i=1}^{k} \exists C_i \subseteq V(G) \Big(\mathbf{IsCycle}(C_i, G - (Z \setminus \{v_i\})) \wedge C_i \cap T \neq \emptyset \Big) \Bigg).\end{aligned}$$

Here, $\mathbf{IsCycle}(C, H)$ is a CMSO formula that expresses that C is a cycle in the subgraph induced by H. We define it explicitly as follows:

$$\begin{aligned}\textbf{IsCycle}(C,H) \equiv (C \neq \emptyset)\\ &\wedge\ \forall u,v \in C:\ E(u,v) \rightarrow (u \in H \wedge v \in H)\\ &\wedge\ \forall u \in C:\ \exists v,w \in C,\ v \neq w \wedge E(u,v) \wedge E(u,w)\\ &\wedge\ \forall u \in C,\ \forall x,y,z \in C:\ (E(u,x) \wedge E(u,y) \wedge E(u,z))\\ &\qquad\qquad \rightarrow (x = y \vee y = z \vee x = z)\\ &\wedge\ \textbf{conn}(C).\end{aligned}$$

This formula ensures that:

- Every vertex in C is contained in H.
- Each vertex in C has exactly two neighbors in C, enforcing the cycle condition.
- The set C is connected in H, ensuring that it forms a single cycle.

It is clear that the size of the above formula ψ depends only on k. □

Because of Lemma 97 we can invoke Proposition 74. We invoke Proposition 74 with $c = 2k$. Let q be the s from this proposition that corresponds to this choice of c. We conclude that to prove Theorem 77 it is enough to prove Theorem 80.

Theorem 80. *For any positive integers $q, k \geq 1$,* MaxMin Subset FVS *on (q,k)-unbreakable graphs on n vertices can be solved in time $2^{(qk)^{\mathcal{O}(q)}} \cdot n^{\mathcal{O}(1)}$.*

We prove Theorem 80 in Section 6.4.1. Next we define a certificate for minimality of subset feedback vertex set for a vertex set V'. The existence of such certificates guarantees the existence of a minimal T-subset feedback vertex set which contains V'.

Definition 31. *Given a graph G and a set of vertices $V' \subseteq V(G)$, we say that an induced subgraph G' of G is a certificate for the T-subset feedback vertex set-minimality of V', if no cycle in G' intersects T and for every $v \in V'$, $G[V(G') \cup \{v\}]$ contains a cycle containing at least one vertex from T.*

Lemma 98. *Given a graph G and a set of vertices V', if there is a certificate for the T-subset feedback vertex set-minimality of V' then there exists a minimal T-subset feedback vertex set of G that contains all the vertices in V'.*

Proof. Let G' be a certificate for minimality of V'. First observe that $V' \cap V(G') = \emptyset$ because no cycle in G' intersects T, but $G[V(G') \cup \{v\}]$, for any $v \in V'$, contains a cycle containing at least one vertex from T. We now construct a minimal T-subset feedback vertex set of G, say Z, iteratively as follows. Initialize $Z = \emptyset$. Fix an arbitrary ordering of the vertices in $V(G) \setminus V(G')$ (note that $V' \subseteq V(G) \setminus V(G')$). Traverse the vertices of $V(G) \setminus V(G')$ in this order. For any vertex $v \in V(G) \setminus V(G')$ in the chosen order:

- if no cycle in $G[V(G') \cup \{v\}]$ intersects T, update $G' := G[V(G') \cup \{v\}]$, that is add v to G', otherwise
- $G[V(G') \cup \{v\}]$ contains contains a cycle containing at least one vertex from T, in which case add v to the set Z.

When all the vertices in $V(G) \setminus V(G')$ have been processed as stated above, then the set Z is a minimal T-subset feedback vertex set of G. Moreover, one can observe that if we start with G' which a certificate for minimality of V' then $V' \subseteq Z$. □

6.4.1 MaxMin Subset Feedback Vertex Set on $(q, 2k)$-unbreakable graphs

The goal of this section is to prove Theorem 80. Let (G, T, k) be an instance of MaxMin Subset Feedback Vertex Set. A vertex $v \in T$ is called *irrelevant* if (G, T, k) is equivalent to $(G - v, T \setminus \{v\}, k)$. To prove Theorem 80, it is enough to prove Theorem 99.

Theorem 99. *For positive integers $q, k \geq 1$, given as input a graph G which is $(q, 2k)$-unbreakable on at least $(2q+1)(2q+2) \cdot (2q+2)! \cdot (k-1)^{2q+2}$ vertices, there exists an algorithm that runs in time $(qk)^{\mathcal{O}(q)} \cdot n^{\mathcal{O}(1)}$, and either returns a minimal T-subset feedback vertex set of G of size at least k or, outputs an irrelevant vertex v.*

To see the proof of Theorem 80 assuming Theorem 99, observe that if the number of vertices is at most $(2q+2)^2(2q+2)!(k-1)^{2q+2}$, then the problem can be solved using brute-force. Otherwise, the algorithm of Theorem 99 either reports a yes instance, or finds an irrelevant vertex v, in which case, delete v from the graph and solve the problem on $G-v$ (which is not necessarily $(q,2k)$-unbreakable). The rest of the section is devoted to the proof of Theorem 99.

Observation 6.4.1. *Let (G,T,k) be a* MaxMin Subset Feedback Vertex Set *instance. If there exists a vertex $v \in V(G)$ that does not participate in any induced cycle of G intersecting T then v is an irrelevant vertex. That is, the instance (G,T,k) is equivalent to the instance $(G-v, T \setminus \{v\}, k)$.*

Proof. To prove a vertex $v \in V(G)$ is irrelevant vertex we will have to show that (G,T,k) is equivalent to $(G-v, T\setminus\{v\}, k)$. For the forward direction, let Z be a minimal T-subset fvs in G of size at least k. Then $G-Z$ does not contain a cycle containing T. Additionally, for any $z \in Z$, the graph $G-(Z\setminus\{z\})$ contains a cycle which goes through z and intersect T, implying the existence of a cycle C_z containing z such that $V(C_z)\cap T \neq \emptyset$. As the cycle is an induced cycle, we know that the vertex v does not lie on this cycle. We will show that Z is a minimal $(T\setminus\{v\})$-subset fvs in $G-v$. Clearly, $G-(Z\cup\{v\})$ does not contain a cycle intersecting $T\setminus\{v\}$. For any vertex $z\in Z$, there is an induced cycle C_z intersecting $T\setminus\{v\}$ in $G-(Z\cup\{v\})$. Therefore, Z is also a minimal $(T\setminus\{v\})$-subset fvs in $G-v$.

For the other direction, let Z be a minimal $(T\setminus\{v\})$-subset fvs in $G-v$ with $|Z|\geq k$. Then, $G-(Z\cup\{v\})$ does not contain a cycle intersecting $T\setminus\{v\}$. For each $z\in Z$, the graph $G-((Z\setminus\{z\})\cup\{v\})$ contains an induced cycle C_z intersecting $T\setminus\{v\}$. Since v is not part of any induced cycle intersecting T, $G-(Z\setminus\{z\})$ contain an induced cycle intersecting T. Hence Z is a minimal T-subset fvs in G. □

Lemma 100. *For any positive integers q,k, if G is $(q,2k)$-unbreakable and there exists an induced cycle in G of length at least $2q+2$ containing a vertex from T, then G has a minimal T-subset feedback vertex set of size at least k.*

Proof. Let C be an induced cycle of length at least $2q+2$ in G containing a vertex, say t, from T. Let $x,y \in V(C)\setminus\{t\}$ be arbitrarily chosen vertices such that $C\setminus\{x,y\}$ contains exactly two paths P_1 and P_2 each of length at least q each and either $P_1\cap T\neq\emptyset$ or $P_2\cap T\neq\emptyset$. Let $Z_x := \{x\}$ and $Z_y := \{y\}$.

Below we define a procedure that iteratively grows the sets in (P_1, P_2, Z_x, Z_y) while maintaining the following invariants.

- The sets P_1, P_2 and $Z_x \cup Z_y$ are pairwise disjoint.
- $G[P_1]$ is connected and does not contain any cycle that contains a vertex from set T and $|P_1| \geq q$.
- $G[P_2]$ is connected and does not contain any cycle that contains a vertex from set T and $|P_2| \geq q$.
- $G[P_1 \cup P_2 \cup \{y\}]$ is a certificate for the T-subset feedback vertex set-minimality for Z_x (and in particular, $G[P_1 \cup P_2 \cup \{y\}]$ does not contain any cycle that contains a vertex from set T).
- $G[P_1 \cup P_2 \cup \{x\}]$ is a certificate for the T-subset feedback vertex set-minimality for Z_y (and in particular, $G[P_1 \cup P_2 \cup \{x\}]$ does not contain any cycle that contains a vertex from set T).
- $N(x) \cap P_1 \neq \emptyset$, $N(x) \cap P_2 \neq \emptyset$, $N(y) \cap P_1 \neq \emptyset$ and $N(y) \cap P_2 \neq \emptyset$.

Observe that the starting sets (P_1, P_2, Z_x, Z_y) defined earlier satisfy these invariants. Note that in invariants 4 and 5, the sets $G[P_1 \cup P_2 \cup \{y\}]$ and $G[P_1 \cup P_2 \cup \{x\}]$, which serve as a certificate of minimality for Z_x and Z_y respectively, rely on the fact that C is an induced cycle. Since the iterative procedure grows these sets, the last property always hold. Also $G[P_1 \cup P_2 \cup \{x\} \cup \{y\}]$ contains a cycle which contains a vertex from T. The idea is to grow these sets until Z_x or Z_y has size at least k. If this happens, then we can use Lemma 98, to conclude that G has a minimal T-subset feedback vertex set of size at least k.

Since G is $(q, 2k)$-unbreakable and $|P_1|, |P_2| \geq q$, we have that every P_1P_2-separator in G has size at least $2k+1$. In particular, $|N(P_1) \cup N(P_2)| \geq 2k+1$. Thus, if we guarantee that $(N(P_1) \cup N(P_2)) \subseteq (Z_x \cup Z_y)$, then $|Z_x \cup Z_y| \geq 2k+1$. Hence either $|Z_x| \geq k$ or $|Z_y| \geq k$.

Towards this we grow the sets in (P_1, P_2, Z_x, Z_y) as follows. Let us call the vertices in $P_1 \cup P_2 \cup Z_x \cup Z_y$ as marked.

Claim 6.4.1. *Let $v \in N(P_1) \cap N(P_2)$ be an unmarked vertex. Either $G[P_1 \cup P_2 \cup \{y, v\}]$ or $G[P_1 \cup P_2 \cup \{x, v\}]$ contains a cycle containing a vertex from T.*

Proof. Without loss of generality, we assume that $t \in P_2 \cap T$. Let $x_2 \in P_2 \cap V(C) \cap N(x)$ and $y_2 \in P_2 \cap V(C) \cap N(y)$. Let $P_{v,t}$ be a arbitrary path from v to t in $G[P_2 \cup \{v\}]$. The intersection between paths $P_{v,t}$ and $P_{x_2,y_2} \subsetneq C$ is not empty because t belongs to both paths. Let v_2 be the first point of intersection between the path $P_{v,t}$ and $P_{x_2,y_2} \subsetneq C$. Then it is clear that either $v_2 \in P_{x_2,t}$ or $v_2 \in P_{y_2,t}$.

We claim that if $v_2 \in P_{x_2,t} \subsetneq C$, then $G[P_1 \cup P_2 \cup \{y, v\}]$ contains a cycle containing a vertex $t \in T$. To construct the cycle, we consider the following sequence of paths and edges:

1. The subpath of $P_{v,t}$ from v to v_2, which lies entirely within $G[P_2 \cup \{v\}]$.
2. The subpath of the cycle C from v_2 to y_2, lying within $P_{x_2,y_2} \subsetneq C$, since $v_2 \in P_{x_2,t}$ and $y_2 \in P_2 \cap V(C) \cap N(y)$. This subpath contains t.
3. The edge y_2y, which exists by the assumption that $y_2 \in N(y)$.
4. A path from y to v, which lies in $G[P_1 \cup \{v, y\}]$. This type of path exists because $G[P_1]$ is connected and $v, y \in N(P_1)$.

Similarly, if $v_2 \in P_{y_2,t} \subsetneq C$, then $G[P_1 \cup P_2 \cup \{x, v\}]$ contains a cycle containing a vertex $t \in T$. ▷

Let v be an arbitrarily chosen unmarked vertex.

Case 1: $v \in N(P_1) \cap N(P_2)$**.** From Lemma 6.4.1, either $G[P_1 \cup P_2 \cup \{x, v\}]$ or $G[P_1 \cup P_2 \cup \{y, v\}]$ or both contain a cycle containing a vertex from T. If $G[P_1 \cup P_2 \cup \{x, v\}]$ contains a cycle containing a vertex from T, then update $Z_y := Z_y \cup \{v\}$. If $G[P_1 \cup P_2 \cup \{y, v\}]$ contains a cycle containing a vertex from T, then update $Z_x := Z_x \cup \{v\}$. If both are true then update $Z_x := Z_x \cup \{v\}$ and $Z_y := Z_y \cup \{v\}$. The sets P_1, P_2 remain the same. Observe that in either case, the updated sets satisfy all the invariants.

Case 2: $v \in N(P_1) \setminus N(P_2)$**.** In this case, if $G[P_1 \cup \{v\}]$ does not contain a cycle containing a vertex from T, then update $P_1 := P_1 \cup \{v\}$. The other sets P_2, Z_x, Z_y remain the same. Note that the updated sets (in particular P_1) maintain the invariant. Otherwise $G[P_1 \cup \{v\}]$ contains a cycle containing a vertex from T. In this case update $Z_x := Z_x \cup \{v\}$ and $Z_y := Z_y \cup \{v\}$.

Case 3: $v \in N(P_2) \setminus N(P_1)$**.** This case is symmetric to Case 2.

We repeat the above process until we have sets (P_1, P_2, Z_x, Z_y) such that all vertices in $N(P_1) \cup N(P_2)$ are marked. In particular, in this case $(N(P_1) \cup N(P_2)) \subseteq (Z_x \cup Z_y)$. As argued earlier, this implies either $|Z_x| \geq k$ or $|Z_y| \geq k$. In both cases, we report that G has a minimal T-subset feedback vertex set of size at least k from Lemma 98. □

Lemma 101. *Let (G, T, k) be an instance of* MaxMin Subset Feedback Vertex Set. *Let d be any positive integer. If $\mathcal{F}$ is a family containing distinct induced cycles intersecting T of G of length at most d and $|\mathcal{F}| > d(d!)(k-1)^d$ then G has a minimal T-subset fvs of size at least k.*

Proof. From the Sunflower Lemma (Theorem 75), we can conclude that there exist at least k induced cycles $\{F_1, F_2, \ldots, F_k\} \subseteq \mathcal{F}$ such that $V(F_i) \cap V(F_j) = Y$ for all $i, j \in [k]$. As the cycles in $\mathcal{F}$ are induced cycles, the graph induced by the set of vertices in Y must be acyclic. Note that Y could possibly be empty. We claim that G has a minimal T-subset fvs of size at least k. Such a minimal T-subset fvs can be obtained by the greedy algorithm described in the proof of Lemma 98. We start with the induced subgraph $G[Y]$. Clearly, any minimal T-subset fvs obtained by this algorithm will contain at least one vertex from $V(F_i) \setminus Y$ for each $i \in [k]$. Since the sets $V(F_i) \setminus Y$ for each $i \in [k]$ are disjoint, the minimal T-subset fvs must have a size of at least k. □

Lemma 102. *Given a graph G, and positive integers d, k, there is an algorithm that runs in $(qk)^{\mathcal{O}(q)} \cdot n^{\mathcal{O}(1)}$ time, and correctly outputs one of the following:*

1. *an induced cycle of length at least d intersecting T,*
2. *a family $\mathcal{F}$ of distinct induced cycles intersecting T, each of length at most $d-1$, such that $|\mathcal{F}| \geq d \cdot d! \cdot (k-1)^d$,*
3. *a set of all the irrelevant vertices in graph G.*
4. *$|V(G)| \leq (d-1)d \cdot d! \cdot (k-1)^d - 1$*

Proof. We design a branching algorithm with a branching factor and depth bounded by a function of d and k. We input a family $\mathcal{F}$ of induced cycles intersecting T of

length at most $d-1$ in G. In the beginning, we take $\mathcal{F} = \emptyset$. In each of the iteration, we either find an induced cycle intersecting T distinct from all the induced cycles in $\mathcal{F}$ or output $\mathcal{F}$ as a family of all the induced cycles intersecting T of length at most $d-1$ in G.

Case 1: Let us assume that there exists an induced cycle F' intersecting T that is distinct from all the cycles in $\mathcal{F}$. We guess the intersection of F' with each cycle in $\mathcal{F}$ and delete the vertices which are not in F but are there in $V(\mathcal{F}) = \bigcup_{F \in \mathcal{F}} V(F)$. Assuming the guess is correct, two possibilities arise. Either the newly found induced cycle F' is of length at least d in which case, we can just return this cycle or the length of this cycle is at most $d-1$ in which case we update $\mathcal{F} := \mathcal{F} \cup \{F'\}$. Note that we iterate this algorithm at most $d \cdot d! \cdot (k-1)^d$ times as after this iteration we have $|\mathcal{F}| \geq d \cdot d! \cdot (k-1)^d$. Note that since all the cycles in $\mathcal{F}$ are of size at most $d-1$, the branching factor at the ith iteration is at most $2^{(d-1)i}$. As we are doing at most $d \cdot d! \cdot (k-1)^d$ many iterations, the depth of the branching tree is bounded by a function of d and k.

Case 2: If there is no induced cycle F' intersecting T distinct from cycle in $\mathcal{F}$ then in all the guesses we will fail to find a new induced cycle intersecting T. In this case, we return $\mathcal{F}$ as a family of all the induced cycles intersecting T of length at most $d-1$ in G. Note that if at some iteration, we get $\mathcal{F}$ as a family of all the induced cycles intersecting T of length at most $d-1$ in G then any vertex that is not in $V(\mathcal{F})$ is irrelevant. One can see that if there is no irrelevant vertex then $V(G) = V(\mathcal{F})$. Therefore, we get $|V(G)| = |V(\mathcal{F})| < (d-1)d \cdot d! \cdot (k-1)^d$. □

Proof of Theorem 99. The algorithm for Theorem 99 proceeds as follows. We run Lemma 102 on input $(G, T, 2q+2, k)$. As we are assuming that $|V(G)| \geq (2q+1)(2q+2)) \cdot (2q+2)! \cdot (k-1)^{2q+2}$, the Lemma 102 must output one of the first three conditions. In the first and second condition, we can return a minimal T-subset feedback vertex set of size at least k due to Lemma 100 and Lemma 101 respectively. If condition three is returned, we get a set of all irrelevant vertices. Then we can delete them from G and solve the problem on the reduced graph. This is safe because of Observation 6.4.1. This finishes the proof of Theorem 99. If the fourth condition is outputted then we can solve the problem by brute force in FPT time. □

6.5 MaxMin List Allocation

In this section, we deal with the MaxMin List Allocation problem.

Definition 32. *Given an instance I of* MaxMin List Allocation*, we say that $A \subseteq V(G)$ is a certificate of correctness for instance I if there exists a partition of A into r non-empty parts $A_1, A_2, \ldots, A_r$ such that in the graph $G[A]$, we have $|E(A_i, A_j)| \geq \alpha_{ij}$ for all $1 \leq i < j \leq r$. The size of the certificate A is $|A|$.*

Lemma 103. *If I is a yes instance, then there exists a certificate of correctness for instance I of size at most $\binom{r}{2}\alpha_{\max}$.*

Proof. Let I be a yes instance. This implies that there exists a partition of $V(G)$ into r non-empty parts $V_1, V_2, \ldots, V_r$ that respects the function λ such that $|E(V_i, V_j)| \geq \alpha_{ij}$ for all $1 \leq i < j \leq r$. Note that for every pair V_i, V_j, we can arbitrarily choose α_{ij} edges between them and add their endpoints to set A. It is clear that due to construction of set A, it forms a certificate of correctness for instance I and the size is bounded by $\binom{r}{2}\alpha_{\max}$. □

Definition 33. *Let $\chi : V(G) \to [r]$ be a coloring of the vertices of graph G with r colors. We say that the coloring given by χ respects the function λ if $\chi(v) \in \lambda(v)$.*

Let $\chi : V(G) \to [r]$ be a coloring of $V(G)$, chosen uniformly at random that respects the function λ, that is, each element v of $V(G)$ is colored with one of the $\lambda(v)$ colors with probability $|\lambda(v)|^{-1}$. Given a coloring χ, we say that a certificate of correctness for instance I, assuming it exists, is r-colorful if $\chi(v) = i$ for all $v \in A_i$ and $i \in [r]$. We say that χ is a good coloring if some certificate of correctness for instance I is r-colorful.

Lemma 104. *Given an instance I, and a coloring $\chi : V(G) \to [r]$ chosen uniformly at random that respects the function λ is good with probability at least $\left(\frac{1}{r}\right)^{\binom{r}{2}\alpha_{\max}}$.*

Proof. Let us assume that the given instance I is a yes instance. Due to Lemma 103, there exists a certificate of correctness for instance I of size at most $\binom{r}{2}\alpha_{\max}$. Note that an element v of A, that is in A_i is colored with color i by probability at least $|\lambda(v)|^{-1} \geq \frac{1}{r}$. Therefore, the probability that the certificate of correctness A is r-colorful with probability at least $r^{-\binom{r}{2}\alpha_{\max}}$. □

Given a coloring of $V(G)$, chosen uniformly at random that respects the function λ, it can be checked in $n^{\mathcal{O}(1)}$ if it is a good coloring. Also it returns a certificate of correctness for instance I of size at most $\binom{r}{2}\alpha_{\max}$ in the same time. Following the standard techniques, we provide the following lemma.

Theorem 78. *There exists a randomized algorithm that, given an instance of* MaxMin List Allocation*, in time* $r^{\binom{r}{2}\alpha_{\max}}.n^{\mathcal{O}(1)}$ *either reports a failure or finds a* r*-colorful certificate of correctness of the given instance. Moreover, if the algorithm is given a yes instance, it returns a solution with constant probability.*

Proof. We show an algorithm that runs in time $r^{\binom{r}{2}\alpha_{\max}}.n^{\mathcal{O}(1)}$, and, given a yes-instance, returns a solution with probability at least $r^{r-\binom{r}{2}\alpha_{\max}}$ due to Lemma 104. Clearly, by repeating the algorithm independently $r^{\binom{r}{2}\alpha_{\max}-r}$ times, we obtain the running time bound and success probability guarantee promised by the theorem statement.

Given an input instance, we uniformly at random color the vertices of $V(G)$ that respects the λ function. That is, every vertex v is colored independently with one color from the set $\lambda(v)$ with uniform probability. Denote the obtained coloring by $\chi : V(G) \to [r]$. We can check in polynomial time whether we have a good coloring. If yes, then we return this partition as a solution. Otherwise, we report failure.

Clearly, any partition returned by the algorithm satisfies the required conditions. It remains to bound the probability of finding a partition in the case input is a yes-instance.

To this end, suppose G has a certificate of correctness on at most $\binom{r}{2}\alpha_{\max}$ vertices due to Lemma 103. By Lemma 104, certificate of correctness becomes r-colorful in the coloring χ with probability at least $r^{r-\binom{r}{2}\alpha_{\max}}$. If this is the case, the algorithm finds an r-colorful certificate of correctness, and the algorithm returns a valid partition of $V(G)$. This concludes the proof of the theorem. □

Theorem 105. *The* MaxMin List Allocation *is W[1]-hard parameterized by* $\alpha_{\max}$ *even when*

- *input graph is bipartite,*
- $\lambda(v) = \{1, 2, \ldots, r\}$ *for all* $v \in V(G)$*,*

- *all but one α_{ij} is zero, that is, $\alpha_{\max} = \sum_{1 \le i < j \le r} \alpha_{ij}$ and*
- *each $G[V_i]$ is an independent set.*

Proof. To prove the W[1]-hardness of the MaxMin List Allocation parameterized by $\alpha_{\max}$, we provide a parameterized reduction from k-Biclique to MaxMin List Allocation. It is known that k-Biclique is W[1]-hard even on bipartite graphs. Given a graph G and an integer k, the k-Biclique asks whether G contains a complete bipartite subgraph with k vertices on each side. Given an instance $I = (G, k)$ of the k-Biclique with $G = (A \cup B, E)$, we construct an instance $I' = (G', r, \lambda, (\alpha_{i,j})_{1 \le i < j \le r})$ of MaxMin List Allocation. Note that we can assume that $|A| = |B| = n$ by adding required number of vertices in G. We define $G' = G$, $r = 2(n-k)+2$, $\lambda(v) = \{1, 2, \ldots, r\}$ $\forall v \in V(G)$ and $\alpha_{ij} = k^2$ when $(i, j) = (1, 2)$, $\alpha_{ij} = 0$ otherwise. Now we prove that $I = (G, k)$ is a yes instance of k-Biclique if and only if $I' = (G, r, \lambda, (\alpha_{ij})_{1 \le i < j \le r})$ is a yes-instance of MaxMin List Allocation.

Let $I = (G, k)$ be a yes instance of k-Biclique. This means that there exists a biclique $K_{k,k}$ as a subgraph of G. Note that $|V(G)| - |V(K_{k,k})| = 2(n-k)$. We arbitrarily enumerate the vertices in the elements of $V(G) \setminus V(K_{k,k})$ as $v_3, v_4, \ldots, v_r$. We give an r-allocation $\mathcal{V} = \{V_1, V_2, \ldots, V_r\}$, where $V_1 = V(K_{k,k}) \cap A$, $V_2 = V(K_{k,k}) \cap B$ and $V_i = \{v_i\}$ $\forall i \in [r] \setminus \{1, 2\}$. It is easy to see that $V_i \neq \emptyset$ for all $i \in [r]$. Since $K_{k,k}$ is a biclique, we have $|E(V_1, V_2)| = k^2$. Hence $|E(V_1, V_2)| \ge \alpha_{12}$. $E(V_i, V_j)| \ge \alpha_{ij}$ is obviously true when $(i, j) \neq (1, 2)$. Since A and B are independent sets and $V_1 \subseteq A, V_2 \subseteq B$, $G[V_1]$ and $G[V_2]$ are independent sets. Also since V_i is a singleton set for $i \in [r] \setminus \{1, 2\}$, $G[V_i]$ is an independent set for $i \in [r] \setminus \{1, 2\}$. Hence I' is a yes instance of MaxMin List Allocation.

Let $I' = (G, r, \lambda, (\alpha_{ij})_{1 \le i < j \le r})$ be a yes instance of MaxMin List Allocation. This means there exists an r-allocation, say $\mathcal{V} = \{V_1, V_2, \ldots, V_r\}$ such that $|E(V_i, V_j)| \ge \alpha_{ij}$ for all $1 \le i < j \le r$ and V_i is non-empty for each $i \in [r]$. Since $r = n(k-2)+2$ and V_i is non-empty for each $i \in [r]$, we must have $|V_1| + |V_2| \le 2k$. Suppose $|V_1| = \ell$, then $|V_2| \le 2k - \ell$. Note that $|E(V_1, V_2)|$ can be at most $\ell(2k - \ell)$. We also know that $|E(V_1, V_2)| \ge k^2$. It is easy to see that $\ell(2k - \ell)$ is at least k^2 only when $\ell = k$. Hence we have $|V_1| = |V_2| = k$. Note that $|E(V_1, V_2)| \ge k^2$ is possible only when every vertex of V_1 is adjacent to every vertex of V_2. Therefore $G[V_1 \cup V_2]$ is k-biclique in G. Hence I is a yes instance of k-Biclique. □

6.6 Conclusions and Open Problems

In this chapter, we studied the parameterized complexity of three MaxMin separation problems and established a range of complexity results. While problems like MaxMin Multiway Cut-Uncut, MaxMin Multiway Separator-Unseparator and MaxMin List Allocation are para-NP-hard or W[1]-hard under natural parameters, we identified FPT results under combined or restricted settings. Although we established fixed-parameterized tractability in various cases, developing more efficient or explicit FPT algorithms remains an open and significant challenge.

Chapter 7

Classical Complexity of Vertex Splitting Problems

A *vertex split* is a graph modification in which a vertex v is replaced by two new vertices, v_1 and v_2. Each edge originally incident to v is reassigned to either v_1 or v_2.

Abu-Khzam et al. [3] proposed two types of vertex splitting operations:

- *Inclusive Vertex Splitting:* $N(v_1) \cup N(v_2) = N(v)$.
- *Exclusive Vertex Splitting:* $N(v_1) \cup N(v_2) = N(v)$ and $N(v_1) \cap N(v_2) = \emptyset$.

The key difference between these two operations lies in the treatment of vertex neighborhoods. In Inclusive Vertex Splitting, v_1 and v_2 are allowed to share neighborhoods, whereas in Exclusive Vertex Splitting, their neighborhoods must be disjoint.

Vertex Splitting was originally used in the context of graph planarization, but now it is applied in a variety of settings. For each fixed graph class $\mathcal{F}$, we define a distinct vertex splitting problem.

$\mathcal{F}$-Vertex Splitting ($\mathcal{F}$-VS)
Input: An undirected graph $G = (V, E)$ and a positive integer k.
Question: Is there a sequence of at most k vertex splits that transforms G into a graph G' such that $G' \in \mathcal{F}$?

Now we develop the theory of Γ-decomposition, which will be helpful for explaining the reductions in this section.

Let Γ be a family of graphs. A Γ-decomposition of a graph G is a set $\{H_1, H_2, \ldots, H_l\}$ of subgraphs of G such that the sets $E(H_1), \ldots, E(H_l)$ form a partition of $E(G)$, and each H_i is isomorphic to a graph of Γ. If Γ contains only one graph, say H, we refer to a Γ-decomposition as an H-decomposition for simplicity.

Since Γ-decomposition partitions the edge set of a graph, it is commonly assumed in most contexts, that Γ contains only connected graphs. Throughout this work, whenever we refer to a Γ-decomposition, we assume that Γ is a collection of connected graphs. Let $\{H_1, H_2, \ldots, H_l\}$ be a Γ-decomposition of G, where each H_i is connected. In such cases, knowing the edge set of H_i suffices to determine its vertex set, since the vertices are simply the endpoints of the edges in H_i. Therefore, it is convenient to describe each H_i by its edge set alone. Let $E_i = E(H_i)$ for each $1 \leq i \leq l$. Then, for notational convenience, we write $\{E_1, E_2, \ldots, E_l\}$ instead of $\{H_1, H_2, \ldots, H_l\}$.

Formally, we assume that all graphs under consideration have no isolated vertices. This assumption implies that every vertex of G must be included in at least one subgraph of the decomposition.

Definition 34. *For a Γ-Decomposition $\mathcal{P} = \{H_1, H_2, \ldots, H_l\}$ of G, the weight of $\mathcal{P}$ is defined as*

$$\mathtt{wgt}_G(\mathcal{P}) = \sum_{v \in V} \#\mathcal{P}(v),$$

where $\#\mathcal{P}(v) := |\{i \mid v \in V(H_i) \in \mathcal{P}\}|$ counts the number of sets in $\mathcal{P}$ that contain the vertex v.

If G is a graph with no isolated vertices, then one can observe that

$$\mathtt{wgt}_G(\mathcal{P}) = \sum_{v \in V} \#\mathcal{P}(v) = \sum_{1 \leq i \leq l} |V(H_i)|.$$

Lemma 106. *Let $G = (V, E)$ be a graph without isolated vertices, and let $\mathcal{P}$ be a Γ-decomposition of G such that $\mathtt{wgt}_G(\mathcal{P}) \leq |V(G)| + \alpha$. Then, either G is already a disjoint union of graphs of Γ or there exists a vertex $u \in V$ such that u can be exclusively split in G to obtain a graph G' satisfying:*

1. *G' admits a Γ-decomposition, say $\mathcal{P}'$,*

2. *$\mathtt{wgt}_{G'}(\mathcal{P}') \leq |V(G')| + \alpha - 1$,*

3. *G' has no isolated vertices.*

Proof. Suppose $\mathcal{P} = \{H_1, H_2, \ldots, H_l\}$ is a Γ-decomposition of G. If G is not already a disjoint union of graphs of Γ, there must exist H_i and H_j such that $V(H_i) \cap V(H_j) \neq \emptyset$. Without loss of generality, we assume $H_i = H_1$ and $H_j = H_2$. In this case, let $u \in V(H_1) \cap V(H_2)$.

We define G' as the graph obtained by splitting u into two vertices u_{in} and u_{out} as follows:

$$N_{G'}(u_{\text{in}}) := N_G(u) \cap V(H_1),$$
$$N_{G'}(u_{\text{out}}) := N_G(u) \setminus V(H_1).$$

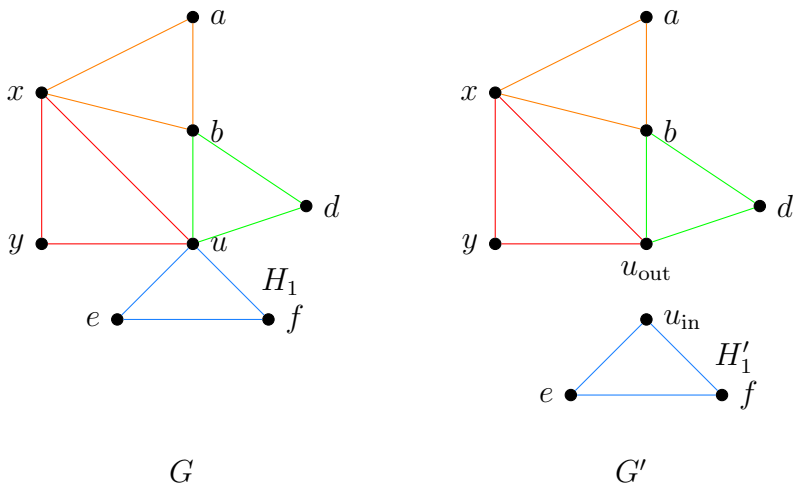


Figure 7.1: On the left: a graph G with a Γ-decomposition where $\Gamma = \{K_3\}$. The subgraph H_1 is shown in blue. On the right: the graph G', obtained by splitting u into u_{in} and u_{out}, along with a corresponding Γ-decomposition of G'.

We construct a Γ-decomposition of G', denoted by $\mathcal{P}' = \{H'_1, H'_2, \ldots, H'_l\}$, where:

$$H_i' := \begin{cases} H_{\text{in}} & \text{if } H_i = H_1 \\ H_{\text{out},i} & \text{if } u \in V(H_i) \wedge (H_i \neq H_1) \\ H_i & \text{otherwise} \end{cases}$$

Here, we define

$$H_{\text{in}} = ((V(H_1) \setminus \{u\}) \cup \{u_{\text{in}}\}, \{u_{\text{in}}v \mid uv \in E(H_1)\} \cup \{xy \mid xy \in E(H_1), (x \neq u) \wedge (y \neq u)\})$$

$$H_{\text{out},i} = ((V(H_i) \setminus \{u\}) \cup \{u_{\text{out}}\}, \{u_{\text{out}}v \mid uv \in H_i\} \cup \{xy \mid xy \in E(H_i), (x \neq u) \wedge (y \neq u)\})$$

Informally, H_{in} is the same as H_1, except the vertex u is relabelled as u_{in}.

We now verify that $\mathcal{P}' = \{H_1', H_2', \ldots, H_l'\}$ is indeed a Γ-decomposition of G'. We first verify that each H_i' is isomorphic to H_i, and hence belongs to Γ.

For H_1, define the map $\phi_1 : H_1 \to H_1'$ by:

$$\phi(v) = \begin{cases} u_{\text{in}} & \text{if } v = u \\ v & \text{otherwise} \end{cases}$$

Clearly, ϕ_1 is an isomorphism from H_1 to H_1', hence $H_1' \cong H_1$.

For any $i \neq 1$ such that $u \in V(H_i)$, define $\phi_{\text{out,i}} : H_i \to H_i'$ by:

$$\phi_{\text{out,i}} = \begin{cases} u_{\text{out}} & \text{if } v = u \\ v & \text{otherwise} \end{cases}$$

This defines an isomorphism from H_i to H_i', and thus $H_i' \cong H_i$.

For all i such that $u \notin V(H_i)$, we have $H_i = H_i'$, so trivially $H_i' \cong H_i$.

Now we have to verify that all the edges of G' are covered by $\mathcal{P}'$. Note that the splitting operation does not change the total number of edges. Since each $H_i' \cong H_i$, we have $|E(H_i)| = |E(H_i')|$. The construction ensures that the H_i' are edge-disjoint. Therefore, the total number

of edges covered by $\mathcal{P}'$ is:

$$\sum_i |E(H_i')| = \sum_i |E(H_i)| = |E(G)| = |E(G')|.$$

Thus, $\mathcal{P}'$ is a valid Γ-decomposition of G'.

We now examine the weight of the decomposition $\mathcal{P}'$ and prove Condition 2. We have

$$\begin{aligned}
\mathtt{wgt}(\mathcal{P}') &= \sum_{v \in V(G')} \#\mathcal{P}'(v) \\
&= \sum_{v \in V(G') \setminus \{u_{\text{in}}, u_{\text{out}}\}} \#\mathcal{P}'(v) + \#\mathcal{P}'(u_{\text{in}}) + \#\mathcal{P}'(u_{\text{out}}) \\
&= \sum_{v \in V(G) \setminus \{u_{\text{in}}, u_{\text{out}}\}} \#\mathcal{P}(v) + 1 + (\#\mathcal{P}(u) - 1) \\
&= \sum_{v \in V(G)} \#\mathcal{P}(v) \\
&\leq |V(G)| + \alpha \\
&= |V(G')| + \alpha - 1
\end{aligned}$$

Finally, since G' is obtained from G by splitting a vertex, it follows that if G has no isolated vertices, then G' does not either. □

7.1 Cycle Graph-Vertex Splitting

We say a graph is a *cycle graph* if it is a disjoint union of cycles.

Definition 35. *A cycle decomposition of a graph G is a set of cycles in G such that every edge of G belongs to exactly one cycle.*

We note that if $\Gamma = \{C_3, C_4, C_5, \ldots\}$, where C_l denotes a cycle of length l for all $l \geq 3$, then a Γ-decomposition and a cycle decomposition are the same notions for a graph.

The following known result will be relevant to our discussion:

Lemma 107. *([51, 101]) A graph G has a cycle decomposition if and only if every vertex of G has even degree.*

Now, we formally define our problem.

Cycle Graph-Exclusive Vertex Splitting (Cycle Graph-EVS)
Input: An undirected graph $G = (V, E)$, and a positive integer k.
Question: Is there a sequence of at most k exclusive vertex splittings that transforms G into a cycle graph?

Lemma 108. *If (G, k) is a yes-instance of* Cycle Graph-Exclusive Vertex Splitting*, then every vertex of G has even degree.*

Proof. Let $G_0, \ldots, G_l$ be a splitting sequence such that $G_0 = G$ and G_l is a cycle graph. Since we perform only exclusive vertex splittings, we have

$$d_G(v) = \sum_{v' : v' \text{ is a descendant of } v \text{ in } G_l} d_{G_l}(v').$$

Here, we say v' is a descendant of v if v' is obtained by splitting v. Since G_l is a cycle graph, every vertex in G_l has even degree. Therefore, every vertex in G has even degree. □

By application of Lemma 108, we get the following corollary.

Corollary 7.1.1. *If G has a vertex with odd degree then (G, k) is a no-instance of* Cycle Graph-Exclusive Vertex Splitting *for any $k \geq 0$.*

Theorem 109. *Let G be a graph where every vertex has even degree. then (G, k) is a yes-instance of* Cycle Graph-Exclusive Vertex Splitting *if and only if $k \geq m - n$, where $|V(G)| = n$ and $|E(G)| = m$.*

Proof. For the forward direction, suppose (G, k) is a yes-instance. For the sake of contradiction, assume that $k < m - n$. Let $G_0, \ldots, G_t$ be a splitting sequence such that $G_0 = G$, G_t is a cycle graph and $t \leq k$. So we have $|V(G_t)| = |V(G)| + t \leq n + k < n + (m - n) = m$. Since G_t is a cycle graph, it satisfies $|E(G_t)| = |V(G_t)| < m$. But since we are doing exclusive vertex splitting, the number of edges remains constant throughput the sequence, so $|E(G)| = |E(G_t| = m$, which is a contradiction. Hence $k \geq m - n$.

For the backward direction, let $k \geq m - n$. By assumption, every vertex of G has even degree. By Lemma 107, we have a cycle decomposition, say $\mathcal{P}$, of G. Note that every cycle

contributes 2 to the degree of the vertices it passes through. Hence every vertex with degree $d(v)$ will be part of exactly $\frac{d(v)}{2}$ elements of $\mathcal{P}$. Therefore, we have $\texttt{wgt}(\mathcal{P}) = \sum_{v \in V(G)} \frac{d(v)}{2} = m$. By iteratively applying Lemma 106 $m-n$ number of times, we can obtain the sequences $G_0, \ldots, G_{m-n}$ and $\mathcal{P}_0, \ldots, \mathcal{P}_{m-n}$.

We claim that G_{m-n} is a disjoint union of cycles (i.e. a cycle graph). As a consequence of the applications of Lemma 106, we get $\texttt{wgt}(\mathcal{P}_{m-n}) \leq |V(G_{m-n})|$. By considering the fact that for each vertex $v \in V(G_{m-n})$ there exists $H_i \in \mathcal{P}_{m-n}$ with $v \in V(H_i)$, we derive $\texttt{wgt}(\mathcal{P}_{m-n}) \geq |V(G_{m-n})|$. Thus, we have that $\texttt{wgt}(\mathcal{P}_{m-n}) = |V(G_{m-n})|$ and it follows that G_{m-n} is a disjoint union of cycles. Hence $(G, m-n)$ is a yes-instance of Cycle Graph-Vertex Splitting. So it is obvious that (G, k) is a yes-instance. □

We note that Corollary 7.1.1 deals with graphs that have at least one vertex with odd degree and Theorem 109 deals with those graphs where all vertices have even degrees. Therefore, by combining Corollary 7.1.1 and Theorem 109, we obtain the following final result.

Theorem 110. *The* Cycle Graph-Exclusive Vertex Splitting *is solvable in polynomial time.*

7.2 Star Forest-Vertex Splitting

A disjoint union of stars is known as a *star forest*. In this section, we prove that both Star Forest-Inclusive Vertex Splitting and Star Forest-Exclusive Vertex Splitting are NP-hard. To do so, we establish the following chain of polynomial-time reductions, starting from the classical NP-hard Vertex Cover problem :

Vertex Cover $\leq_P$ Weighted Star Decomposition

$\leq_P$ Star Forest-Inclusive Vertex Splitting

We formally define the Star Forest-Inclusive Vertex Splitting problem below.

Star Forest-Inclusive Vertex Splitting (Star Forest-IVS)
Input: An undirected graph $G = (V, E)$, and a positive integer k.
Question: Is there a sequence of at most k inclusive vertex splitting that tranforms G into a star forest?

If Γ is a collection of all star graphs, we will use the term *Star-decomposition* to refer to a Γ-decomposition. For the sake of completion, we now define the notion of weight for a Star-decomposition.

Definition 36. *For a Star-decomposition $\mathcal{P}$ of G, the weight of $\mathcal{P}$ is defined as*

$$\mathrm{wgt}_G(\mathcal{P}) = \sum_{v \in V} \#\mathcal{P}(v),$$

where $\#\mathcal{P}(v) := |\{P \mid v \in P \in \mathcal{P}\}|$ denotes the number of sets in $\mathcal{P}$ that contain v.

Now, we formulate the associated decision problem:

Weighted Star-Decomposition
Input: An undirected graph $G = (V, E)$, and a positive integer k.
Question: Does there exist a Star-decomposition $\mathcal{P}$ of G such that $\mathtt{wgt}(\mathcal{P}) \leq k$?

Theorem 111. Weighted Star Decomposition *is NP-complete on cubic graphs and also on planar graphs of maximum degree* 3*.*

Proof. It is easy to see that Weighted Star-Decomposition is in NP. It is known that Vertex Cover is NP-complete on cubic graphs and also on planar graphs of maximum degree 3. To prove that Weighted Star-Decomposition is NP-hard on cubic graphs and also on planar graphs of maximum degree 3, we give a reduction from Vertex Cover. We prove that (G, k) is a yes-instance of Vertex Cover if and only if $(G, m + k)$ is a yes-instance of Weighted Star-Decomposition.

For the forward direction, we assume that (G, k) is a yes-instance of Vertex Cover. Suppose $X = \{v_1, v_2, \ldots, v_l\}$ is a vertex cover of G with $l \leq k$. Now, we define $E_1 = \{v_1 w : w \in N(v_1)\}$ and $E_i = \{v_i w : w \in N(v_i) \setminus \{v_1, v_2, \ldots, v_{i-1}\}\}$ for $2 \leq i \leq l$. We claim that $\mathcal{P} = \{E_1, E_2, \ldots, E_l\}$ is a star decomposition with $\mathtt{wgt}(\mathcal{P}) \leq m + k$.

Clearly, each E_i is a star with center vertex v_i. Note that $G - X$ is an independent set so all the edges of graph G are either between X and $G - X$ or between the vertices of X. The way the construction of E_i is done and X is a vertex cover make it clear that $\mathcal{P}$ covers all edges of G. It is also clear that $E_i \cap E_j = \emptyset$ whenever $i \neq j$.

Note that if E_i has p_i edges then the number of vertices in E_i is $p_i + 1$. Hence

$$\texttt{wgt}(\mathcal{P}) = \sum_{1 \leq i \leq l} |V(E_i)| = \sum_{1 \leq i \leq l} (p_i + 1) = \sum_{1 \leq i \leq l} p_i + l = m + l \leq m + k.$$

For the reverse direction, we assume that there exists a star decomposition, say $\mathcal{S}$ of G with $\texttt{wgt}(\mathcal{S}) \leq m+k$. Suppose $\mathcal{S} = \{S_1, S_2, \ldots, S_q\}$. Define $C = \{c_1, c_2, \ldots, c_q\}$ where c_i is center of star S_i. Suppose a star with center c_i contains p_i number of edges then the number of vertices is $p_i + 1$ in that star. Now we see that $\texttt{wgt}(\mathcal{S}) = \sum_{1 \leq i \leq q}(p_i + 1) = \sum_{1 \leq i \leq q} p_i + q = m + q$. Since $\texttt{wgt}(\mathcal{S}) \leq m + k$, we have $q \leq k$.

We claim that $C = \{c_1, c_2, \ldots, c_q\}$ is a vertex cover of G. Let uv be an edge of graph G. Since $\mathcal{S}$ is a sigma star cover of G, uv must be in S_i for some i. Since S_i is a star graph either u or v must be c_i. The edge uv is covered by C because C contains c_i. So we have proved that C is a vertex cover of size at most k, hence (G, k) is a yes-instance of the Vertex Cover. □

Now we aim to prove that Star Forest-Vertex Splitting is NP-hard by giving a polynomial-time reduction from Weighted Star Decomposition. But we first prove a few lemmas.

Lemma 112. *Let $G = (V, E)$ be a graph, and let $G' = (V', E')$ be obtained from G by inclusive vertex splitting of a vertex $u \in V$ into two vertices $v, w \in V'$. If $\mathcal{P}'$ is a star decomposition of G', then there exists a star decomposition $\mathcal{P}$ of G with $\texttt{wgt}(\mathcal{P}) \leq \texttt{wgt}(\mathcal{P}')$.*

Proof. Let $\mathcal{P}' = \{E'_1, E'_2, \ldots, E'_t\}$ be a star decomposition of G'. Then using

$$f(E'_i) := \begin{cases} (E'_i \setminus \{vz, wz \mid z \in V(E'_i)\}) \cup \{uz \mid z \in V(E'_i) \setminus \{v, w\}\} & \text{if } V(E'_i) \cap \{v, w\} \neq \emptyset \\ E'_i & \text{otherwise} \end{cases}$$

we define

$$\mathcal{P} := \{E_1, E_2, \ldots, E_t \mid f(E_i') = E_i \text{ for } 1 \leq i \leq t\}.$$

Note that f gives a bijection over the domain $\mathcal{P}'$. We claim that $\mathcal{P}$ satisfies the conditions of this lemma. First, we establish that $\mathcal{P}$ is a star decomposition of G.

We begin by proving that all $E_i \in \mathcal{P}$ are stars.

Let $E_i \in \mathcal{P}$. Assume $f^{-1}(E_i) = E_i$. Observe that $E_i \cap \{v, w\} = \emptyset$. Clearly, E_i is a star.

Conversely, assume $f^{-1}(E_i) \neq E_i$. Then two cases can occur.

Case 1: $f^{-1}(E_i)$ contains both v and w. By assumption, $f^{-1}(E_i)$ is a star. Suppose $f^{-1}(E_i)$ contains both v and w. By definition of vertex splitting, it follows that v and w can not be the center of the star $f^{-1}(E_i)$. Let r be the center of $f^{-1}(E_i)$. Then we see that E_i is also a star with center r and contains u.

Case 2: $f^{-1}(E_i)$ contains exactly one of v and w. Without loss of generality, we assume that $v \in f^{-1}(E_i)$ and $w \notin f^{-1}(E_i)$. Then $E_i = (E_i' \setminus \{vz \mid vz \in V(E_i')\}) \cup \{uz \mid z \in V(E_i') \setminus \{v\}\}$. It is clear that E_i' is isomorphic to E_i. Hence E_i is a star.

Now, we prove the second property, that is, all edges of G are covered by $\mathcal{P}$. Let $v_1, v_2 \in V(G)$ with $v_1 \neq v_2$.

Case 1: $\{v_1, v_2\} \cap \{u\} = \emptyset$: The edge v_1v_2 is not affected by the split, therefore $v_1v_2 \in E(G')$. Hence there is some $E_i' \in \mathcal{P}'$ such that $v_1v_2 \in E_i'$. Thus $v_1v_2 \in E_i = f(E_i')$.

Case 2: $\{v_1, v_2\} \cap \{u\} \neq \emptyset$: Without loss of generality, assume $v_1 = u$. The definition of vertex splitting tells that either $vv_2 \in E(G')$ or $wv_2 \in E(G')$ must hold. Without loss of generality, assume the former. Since $\mathcal{P}'$ is a star-decomposition of G there exist some $E_j' \in \mathcal{P}'$ such that $vv_2 \in E_j' \in \mathcal{P}'$. Thus, we find that $u, v_2 \in E_j = f(E_j') \in \mathcal{P}$.

Therefore, $\mathcal{P}$ is a sigma Γ-cover of G. We observe that f ranging over $\mathcal{P}'$ does not increase the cardinality of any image it maps. Hence we have, $\mathtt{wgt}(\mathcal{P}) \leq \mathtt{wgt}(\mathcal{P}')$. □

Since exclusive vertex splitting is also inclusive vertex splitting, so above proof works when we do exclusive vertex splitting of u.

Theorem 113. *Let $G = (V, E)$ be a graph. Then (G, k) is a yes-instance of* Star Forest-

Inclusive Vertex Splitting *if and only if* $(G, |V(G)|+k)$ *is a yes-instance of* Weighted Star Decomposition.

Proof. Let (G, k) be a yes-instance of Star Forest-Vertex Splitting, that is, there exists a sequence of at most k vertex splittings in G such that the resulting graph G' is a disjoint union of stars.

Let $G_0, \ldots, G_l$ be a sequence of graphs with $G_0 = G$ and $l \leq k$ such that each graph, except G_0, is obtained from its predecessor via a vertex split, and G_l is a disjoint union of stars. By identifying all connected components of G_l with their vertex sets, we can construct a star decomposition $\mathcal{P}_l$ of G_l with $\texttt{wgt}(\mathcal{P}_l) = |V(G_l)|$. Each split used in the construction of $G_0, \ldots, G_l$ introduces exactly one new vertex, therefore $|V(G_l)| = |V(G)| + l$. Combining this with the fact that $l \leq k$, we derive $\texttt{wgt}(\mathcal{P}_l) \leq |V| + k$. Using the sequence $G_0, \ldots, G_l$ in reverse order, we iteratively apply Lemma 112 l times using $\mathcal{P}_l$ and G_l as base case and obtain $\mathcal{P}_0, \ldots, \mathcal{P}_l$. In particular, it follows that $\mathcal{P}_0$ is a star decomposition of G satisfying $\texttt{wgt}(\mathcal{P}_0) \leq |V(G)| + k$. Thus, $(G, |V(G)| + k)$ is a yes-instance.

Let $(G, |V(G)| + k)$ be a yes-instance of Weighted Star Decomposition, that is, there exists a star decomposition, say $\mathcal{P} = \{H_1, H_2, \ldots, H_p\}$ with $\texttt{wgt}(\mathcal{P}) \leq n + k$. By iteratively applying Lemma 106 for a number of times, call it l, either until star-forest is obtained as a direct result of the lemma, or alternatively, stopping after k iterations, we can obtain the sequences $G_0, \ldots, G_k$ and $\mathcal{P}_0, \ldots, \mathcal{P}_k$.

We shall now verify that G_l must be a disjoint union of stars. As a consequence of the applications of Lemma 106, we get $\texttt{wgt}(\mathcal{P}_k) \leq |V(G_k)|$. By considering the fact that for each vertex $v \in V(G_k)$ there exists $H_i^k \in \mathcal{P}_k$ with $v \in E_i^k$, we derive $\texttt{wgt}(\mathcal{P}_k) \geq |V(G_k)|$. Thus, we have that $\texttt{wgt}(\mathcal{P}_k) = |V(G_k)|$ and it follows that G_k is a disjoint union of stars. Hence (G, k) is a yes-instance of Star Forest-Vertex Splitting. □

Theorem 114. Star Forest-Inclusive Vertex Splitting *is NP-complete on cubic graphs and also on planar graphs of maximum degree* 3.

Proof. It is easy to see that Star Forest-Inclusive Vertex Splitting is in NP, because a certificate of Star Forest-Inclusive Vertex Splitting can be guessed and checked in polynomial time. By Theorem 113 we conclude that deciding (G, k) instance of Star Forest-Inclusive Vertex Splitting is equivalent to deciding the instance

$(G, k - |V(G)|)$ of Weighted Star Decomposition. Since Weighted Star Decomposition is NP-complete on cubic graphs and even on planar graphs of maximum degree 3, Star Forest-Inclusive Vertex Splitting is NP-hard on cubic graphs and also on planar graphs of maximum degree 3. Consequently, we conclude that Star Forest-Inclusive Vertex Splitting is NP-complete on cubic graphs and also on planar graphs of maximum degree 3. □

Note that the proof of Theorem 113 also applies to the case of exclusive vertex splitting. In the forward direction of the proof, we used Lemma 112, and as observed, Lemma 112 works for exclusive vertex splitting as well. Hence, there is no issue in the forward direction of the proof of Theorem 113. In the backward direction, we used Lemma 106, which involves exclusive vertex splitting. Based on this, we now define the following problem

Star Forest-Exclusive Vertex Splitting (Star Forest-EVS)
Input: An undirected graph $G = (V, E)$, and a positive integer k.
Question: Is there a sequence of at most k exclusive vertex splitting that tranforms G into a star forest?

Then, we have the following two results.

Corollary 7.2.1. *Let $G = (V, E)$ be a graph. Then (G, k) is a yes-instance of* Star Forest-Exclusive Vertex Splitting *if and only if $(G, |V(G)| + k)$ is a yes-instance of* Weighted Star Decomposition

Corollary 7.2.2. *The* Star Forest-Exclusive Vertex Splitting *problem is NP-complete on cubic graphs and also on planar graphs of maximum degree* 3.

7.3 Bipartite-Vertex Splitting

In this section, we prove that Bipartite-Vertex Splitting is NP-complete. Firbas and Sorge [58, 60] showed that Bipartite-Vertex Splitting is NP-hard by providing a reduction from 2-Subdivided Cubic Vertex Cover. We establish that Bipartite-Vertex Splitting is NP-complete by presenting a reduction from Bipartite Vertex Deletion. The advantage of our reduction is that it demonstrates the equivalence between Bipartite-Vertex Splitting and Bipartite Vertex Deletion. So, the existence of a kernel-

ization or an FPT algorithm for Bipartite Vertex Deletion implies the existence of the same for Bipartite-Vertex Splitting. Moreover, our proof is comparatively easy to understand.

We first define our problems formally.

Bipartite-Inclusive Vertex Splitting (Bipartite-IVS)
Input: An undirected graph $G = (V, E)$, and a positive integer k.
Question: Is there a sequence of at most k inclusive vertex splittings that transforms G into a bipartite graph?

Bipartite-Exclusive Vertex Splitting (Bipartite-EVS)
Input: An undirected graph $G = (V, E)$, and a positive integer k.
Question: Is there a sequence of at most k exclusive vertex splittings that transforms G into a bipartite graph?

Theorem 115. *Let G be a graph. Then (G, k) is a yes-instance of* Bipartite-Inclusive Vertex Splitting *if and only if (G, k) is a yes-instance of* Bipartite Vertex Deletion.

Proof. For the forward direction, we assume that (G, k) is a yes-instance of Bipartite-Inclusive Vertex Splitting. Let S be the set of vertices of G that are split to obtain a bipartite graph, with $|S| \leq k$. Suppose that applying inclusive vertex splitting to the vertices in S transforms G into a bipartite graph G'. We claim that $G - S$ is a bipartite graph. Note that $G - S$ is an induced subgraph of G' because splitting the vertices in S does not affect the adjacency between the vertices in $G - S$. Since G' is bipartite and $G - S$ is its induced subgraph, it follows that $G - S$ is also bipartite. So we conclude that S is a bipartite vertex deletion set of size almost k. Hence, (G, k) is a yes-instance of Bipartite Vertex Deletion.

For the reverse direction, we assume that (G, k) is a yes-instance of Bipartite Vertex Deletion. Suppose $S = \{v_1, v_2, \ldots, v_l\}$ is a bipartite-vertex deletion set of G with $l \leq k$. Suppose $G - S$ is a bipartite graph with bipartition V_1 and V_2. We treat S as an ordered set.

Now we construct a sequence $G = G_0, G_1, G_2, \ldots, G_l$ where G_l is obtained by splitting of vertex v_i in the graph G_{i-1}. Note that the construction of G_i is done inductively. Now, we will see how the vertex splitting is done to get G_i from G_{i-1}.

For $1 \leq i \leq l$, we split v_i into two vertices v_{i1} and v_{i2} such that

$$N_{G_i}(v_{i1}) = (N_{G_{i-1}}(v_i) \cap V_2) \cup \left\{v_j : v_j \in N_{G_{i-1}}(v_i) \text{ when } j > i\right\}$$

$$N_{G_i}(v_{i2}) = (N_{G_{i-1}}(v_i) \cap V_1) \cup \{v_{j1} : v_{j1} \in N_{G_{i-1}}(v_i)\}$$

We see that $V_1 \cup \{v_{11}, v_{21}, \ldots v_{l1}\}$ and $V_2 \cup \{v_{12}, v_{22}, \ldots v_{l2}\}$ is a bipartition of the graph G_l. Hence, (G, k) is a yes-instance of Bipartite-Inclusive Vertex Splitting. □

It is easy to see that Bipartite-Inclusive Vertex Splitting and Bipartite-Exclusive Vertex Splitting are in NP. Since we know that Bipartite Vertex Deletion (popularly known as Odd Cycle Transversal) is NP-complete, Bipartite-Inclusive Vertex Splitting is NP-hard. We note that in the reverse direction of the proof, all the vertex splittings that were done are exclusive vertex splittings. There is no problem in the forward direction if we allow only exclusive vertex splittings. It is easy to conclude the following results from the above discussion.

Corollary 7.3.1. Bipartite-Inclusive Vertex Splitting *is NP-complete.*

Corollary 7.3.2. *Let G be a graph. Then (G, k) is a yes-instance of* Bipartite-Exclusive Vertex Splitting *if and only if (G, k) is a yes-instance of* Bipartite-Vertex Deletion.

Corollary 7.3.3. Bipartite-Exclusive Vertex Splitting *is NP-complete.*

7.4 Uniform Cycle Graph-Vertex Splitting

A *uniform cycle graph* is a disjoint union of equal-sized cycles. We have seen that Cycle Graph-Exclusive Vertex Splitting is polynomial-time solvable. See Theorem 110. In Cycle Graph-Exclusive Vertex Splitting, we aimed to get a disjoint union of cycles. Now, if we put a restriction that all cycles must be of the same size, then we get Uniform Cycle Graph-Exclusive Vertex Splitting. It turns out that Uniform

Cycle Graph-Exclusive Vertex Splitting is NP-hard. Proving the NP-completeness of Uniform Cycle Graph-Exclusive Vertex Splitting also motivates us to work on the problem considered in the next section.

Let us define our problems formally.

Uniform Cycle Graph-Exclusive Vertex Splitting (UCG-EVS)
Input: An undirected graph $G = (V, E)$, and a positive integer k.
Question: Is there a sequence of at most k exclusive vertex splittings that transforms G into a uniform cycle graph?

Uniform Cycle Graph-Inclusive Vertex Splitting (UCG-IVS)
Input: An undirected graph $G = (V, E)$, and a positive integer k.
Question: Is there a sequence of at most k inclusive vertex splittings that transforms G into a uniform cycle graph?

Theorem 116. Uniform Cycle Graph-Inclusive Vertex Splitting *is NP-complete.*

Proof. It is clear that UCG-IVS is in NP. To show that the problem is NP-hard, we provide a reduction from K_3-Decomposition. The problem asks to determine whether the given graph has a K_3-decomposition. Ian Holyer proved that K_3-Decomposition is NP-hard [78].

Given an instance $I = G(V, E)$ of K_3-Decomposition, we construct an instance $I' = (G', k')$ of Uniform Cycle Graph-Inclusive Vertex Splitting. To construct the graph G', we start with G and add $m - n + 1$ disjoint triangles to it, where $|V(G)| = n$ and $|E(G)| = m$. We set $k' = m - n$. We now prove that the instances I and I' are equivalent.

Suppose I is a yes-instance, that is, there exists a K_3-decomposition of G. Let $\mathcal{P} = \{P_1, P_2, \ldots, P_{\frac{m}{3}}\}$ be a K_3-decomposition of G. If such a decomposition exists, then each vertex in G must have an even degree. For each vertex $v \in V(G)$, the number of triangles in $\mathcal{P}$ that includes v is $l = \frac{d(v)}{2}$. Therefore, we have

$$\mathtt{wgt}(\mathcal{P}) = \sum_{v \in V(G)} \left(\frac{d(v)}{2} - 1 \right) = m - n.$$

We now iteratively apply Lemma 106 with $\Gamma = K_3$ until we obtain a disjoint union of triangles. The total number of vertex splittings required is exactly equal to $m - n$. Note that no operations are needed on the connected components corresponding to the added disjoint triangles in G', as they are already K_3's. Hence $I' = (G', k')$ is a yes-instance of UCG-IVS.

Suppose I' is a yes-instance, that is, there exists a sequence of at most $m - n$ inclusive vertex splittings in G' that results in a uniform cycle graph G''. We first argue that G'' must be a disjoint union of triangles (that is, K_3's). Since G' contains $m - n + 1$ disjoint triangles and only $m - n$ vertex splittings are allowed, at least one of these triangles remains untouched. Because G'' is a uniform cycle graph and it contains at least one triangle, all components of G'' must be triangles. Thus G'' is a disjoint union of K_3's. We can assume that no vertex from the added triangles (that is, vertices in $V(G') \setminus V(G)$) is split.

Claim 7.4.1. *For every edge $u''v'' \in E(G'')$, there exists a unique edge uv in $E(G')$ such that either $u'' = u$ or u'' is a descendant of vertex u, and similarly, either $v'' = v$ or v'' is is a descendant of vertex v.*

Proof. Note that after $m - n$ vertex splittings, the graph G'' can contain at most $n + 3(m - n + 1) + (m - n) = m + 3(m - n + 1)$ many vertices. Since G'' is disjoint union of cliques of size 3, $|E(G'')| = \frac{3|V(G'')|}{3} = |V(G'')| \leq m + 3(m - n + 1) = |E(G')|$. Since G'' is obtained by vertex splitting operations on G', we must have $|E(G'')| \geq |E(G')|$. This implies that $|E(G'')| = |E(G')|$. Therefore, for every edge $u''v'' \in E(G'')$, there exists an unique edge uv in $E(G')$. □

We also note that no two vertices of a triangle in G'' can have the same ancestor. The above claim shows that, for every triangle in G'' there exists a unique corresponding triangle in G'. As $|E(G'')| = |E(G')|$, this gives us a partition of edges of G' where each part induces a K_3. Since G is an induced subgraph of G', G has a K_3-decomposition. □

Note that the above proof also hilds for exclusive vertex splitting, as one can see that in the backward part of the proof, all vertex splittings performed are exclusive.

Corollary 7.4.1. Uniform Cycle Graph-Exclusive Vertex Splitting *is NP-complete.*

7.5 Uniform Bicluster-Vertex Splitting

A *uniform bicluster graph* is a graph in which each connected component is a complete bipartite graph of the same dimension. The Bicluster-Vertex Splitting problem, where the objective is to transform a given graph G into a bicluster graph using at most k vertex splits, remains an open problem [14]. In this chapter, we consider a slightly different problem, where the goal is to obtain a uniform bicluster graph instead of a general bicluster graph. We prove that Uniform Bicluster-Vertex Splitting is NP-complete.

Interestingly, related problems show contrasting complexity results. For instance, Cycle Graph-Exclusive Vertex Splitting is polynomial-time solvable (see Theorem 110), while its uniform version, Uniform Cycle Graph-Exclusive Vertex Splitting is NP-complete (see Section 7.4). For the general graphs, both Cluser-Inclusive Vertex Splitting [59] and Uniform Cluster-Inclusive Vertex Splitting are NP-complete (see Theorem 21). However, in the special case of graphs with isolated vertices, Cluster-Inclusive Vertex Splitting is NP-complete, while Uniform Cluster-Inclusive Vertex Splitting is polynomial-time solvable. These examples highlight that we cannot directly infer the complexity of Uniform Bicluster-Vertex Splitting from that of Bicluster-Vertex Splitting. Therefore, Uniform Bicluster-Vertex Splitting is a problem of independent interest.

We define our problem formally

Uniform Bicluster-Inclusive Vertex Splitting (Uniform Bicluster-IVS)
Input: An undirected graph $G = (V, E)$, and a positive integer k.
Question: Is there a sequence of at most k inclusive vertex splittings that transforms G into a bicluster graph?

Uniform Bicluster-Exclusive Vertex Splitting (Uniform Bicluster-EVS)
Input: An undirected graph $G = (V, E)$, and a positive integer k.
Question: Is there a sequence of at most k exclusive vertex splittings that transforms G into a bicluster graph?

Now, we will prove that Uniform Bicluster-Inclusive Vertex Splitting is NP-complete. The proof technique is very similar to that used in the proof of Theorem 116.

Theorem 117. Uniform Bicluster-Inclusive Vertex Splitting *is NP-complete.*

Proof. It is clear that Uniform Bicluster-Inclusive Vertex Splitting is in NP. To show that the problem is NP-hard, we provide a reduction from $K_{1,2}$-Decomposition. The problem asks to determine whether the given graph has a $K_{1,2}$-decomposition. Dor and Tarsi [44] proved that $K_{1,2}$-Decomposition is NP-hard.

Given an instance $I = G(V, E)$ of $K_{1,2}$-Decomposition, we construct an instance $I' = (G', k')$ of Uniform Bicluster-Inclusive Vertex Splitting. To construct the graph G', we begin with G and add $\left\lfloor \frac{3m}{2} \right\rfloor - n + 1$ disjoint copies of $K_{1,2}$ to it, where $|V(G)| = n$ and $|E(G)| = m$. We then set $k' = \left\lfloor \frac{3m}{2} \right\rfloor - n$. We now prove that the instances I and I' are equivalent.

Suppose I is a yes-instance, that is, there exists a $K_{1,2}$-decomposition of G. Let $\mathcal{P} = \{P_1, P_2, \ldots, P_{\frac{m}{2}}\}$ be a $K_{1,2}$-decomposition of G. Each P_i is isomorphic to $K_{1,3}$ and hence contains exactly 3 vertices. Therefore

$$\texttt{wgt}(\mathcal{P}) = \frac{3m}{2}.$$

We now iteratively apply Lemma 106 up to k' times, either until a disjoint union of $K_{1,2}$ graphs is obtained, or we reach $G_{k'}$, resulting in a sequence of graphs $G_0, \ldots, G_{k'}$ and corresponding decompositions $\mathcal{P}_0, \ldots, \mathcal{P}_{k'}$.

We now verify $G_{k'}$ is a disjoint union of $K_{1,2}$. By applying Lemma 106 k' times, we get $\texttt{wgt}(\mathcal{P}_{k'}) \leq |V(G_{k'})|$. Since each vertex $v \in V(G_{k'})$ is contained in some part $P_i^{k'} \in \mathcal{P}_{k'}$ with $v \in V(P_i^{k'})$, we also have $\texttt{wgt}(\mathcal{P}_{k'}) \geq |V(G_{k'})|$. Combining both inequalities, we have $\texttt{wgt}(\mathcal{P}_{k'}) = |V(G_{k'})|$. This implies that each part of $\mathcal{P}_{k'}$ is disjoint and fully covers the vertex set. Since each part is a $K_{1,2}$, $G_{k'}$ is a disjoint union of $K_{1,2}$. Note that we do not perform any operations on the connected components of G' that were already $K_{1,2}$ subgraphs. Hence (G', k') is a yes-instance.

For the reverse direction, we assume that there exists a sequence of at most $m-n$ inclusive vertex splittings in G' such that the resulting graph G'' is a uniform bicluster graph. We first argue that G'' must be a disjoint union of $K_{1,2}$. This follows from the fact that G' contains $\left\lfloor \frac{3m}{2} \right\rfloor - n + 1$ connected components that are each isomorphic to $K_{1,2}$. Since only $\left\lfloor \frac{3m}{2} \right\rfloor - n$ vertex splittings are allowed, at least one $K_{1,2}$ component remains untouched in G''. Because

G'' is a uniform bicluster graph, all its components must be isomorphic to the same complete bipartite graph, and thus it must be a disjoint union of $K_{1,2}$ components. We may assume that none of the vertices in $V(G') \setminus V(G)$ are split.

Claim 7.5.1. *For every edge $u''v'' \in E(G'')$, there exists a unique edge uv in $E(G')$ where either $u'' = u$ or u'' is a descendant of vertex u and similarly $v'' = v$ or v'' is is a descendant of vertex v.*

Proof. Note that after $\left\lfloor \frac{3m}{2} \right\rfloor - n$ vertex splittings, the graph G'' can contain at most $n + 3\left(\left\lfloor \frac{3m}{2} \right\rfloor - n + 1\right) + \left(\left\lfloor \frac{3m}{2} \right\rfloor - n\right) = 3\left(\left\lfloor \frac{3m}{2} \right\rfloor - n + 1\right) + \left\lfloor \frac{3m}{2} \right\rfloor$ many vertices. Since G'' is disjoint union of $K_{1,2}$, number of components in G'' is at most $\frac{1}{3}\left(3\left(\left\lfloor \frac{3m}{2} \right\rfloor - n + 1\right) + \left\lfloor \frac{3m}{2} \right\rfloor\right)$. Since each component in G'' has exactly two edges, we have $|E(G'')| \leq \frac{2}{3}\left(3\left(\left\lfloor \frac{3m}{2} \right\rfloor - n + 1\right) + \left\lfloor \frac{3m}{2} \right\rfloor\right) = 2\left(\left\lfloor \frac{3m}{2} \right\rfloor - n + 1\right) + \frac{2}{3}\left\lfloor \frac{3m}{2} \right\rfloor \leq 2\left(\left\lfloor \frac{3m}{2} \right\rfloor - n + 1\right) + m = |E(G')|$. Since G'' is obtained by vertex splitting operations on G', we must have $|E(G'')| \geq |E(G')|$. This implies that $|E(G'')| = |E(G')|$. Therefore, for every edge $u''v'' \in E(G'')$, there exists an unique edge uv in $E(G')$. □

The observation that $|E(G'')| = |E(G')|$ implies all the vertex splitting performed are exclusive. Since we are only applying exclusive vertex splitting, no two vertices in a connected component $K_{1,2}$ of G'' can have the same ancestor. This ensures that each connected component $K_{1,2}$ in G'' corresponds uniquely to a $K_{1,2}$ in G'. As $|E(G'')| = |E(G')|$, this gives us a $K_{1,2}$-decomposition of G'. Because G is an induced subgraph of G', it follows that G also admits a $K_{1,2}$-decomposition. □

Note that the above proof also holds for exclusive vertex splitting, as we observed that in the backward part of the proof all the vertex splittings performed are exclusive. Therefore, we have the following corollary.

Corollary 7.5.1. Uniform Bicluster-Exclusive Vertex Splitting *is NP-complete.*

7.6 Conjecture

Alexander proposed the following conjecture (see Conjecture 3.10 in [58]) in his PhD thesis [58]:

Conjecture 7.6.1. *Let* Π *and* Γ *be classes of graphs such that the following conditions hold:*

- *A graph* G *belongs to* Π *if and only if every connected component of* G *belongs to* Γ.
- *For each graph* $G \in \Gamma$, *any graph* G' *obtained from* G *by merging two non-adjacent vertices of* G *is also in* Γ.

Then, Sigma Γ-Cover $\leq_{\mathrm{p}}$ Π-Vertex Splitting.
If additionally $K_2 \in \Gamma$, *and* Γ *is closed under both deletion and addition of universal vertices, then* Node Γ-Cover $\leq_{\mathrm{p}}$ Sigma Γ-Cover.

In this section, we will prove the second part of the conjecture. To prove Cluster-Vertex Splitting is NP-hard, Alexander at el. [59] gave a polynomial-time reduction from Node Clique Cover to Sigma Clique Cover and a polynomial-time reduction from Sigma Clique Cover to Cluster-Vertex Splitting. So, we also believe that this technique can be generalized to work with other vertex splitting problems associated with different graph covering problems. In the thesis [58], it is mentioned that Sigma Clique Cover and Compact Letter Display-Σ are (disregarding minor technical complications involving isolated vertices) equivalent problems. Thus Sigma Γ-Cover may be of independent interest.

We first define some notions.

Definition 37. Let $\mathcal{P}(V)$ denote the power set of V. A family of sets $\mathcal{U} \subseteq \mathcal{P}(V)$ is called a *node* Γ*-cover* if the following conditions hold:

1. for each $U \in \mathcal{U}$, the induced subgraph $G[U]$ belongs to the class Γ.
2. for each vertex $v \in V(G)$, there exists a set $U \in \mathcal{U}$ such that $v \in U$.

The corresponding decision problem is defined as follows:

Node Γ-Cover
Input: An undirected graph $G = (V, E)$, and a positive integer k.
Question: Does there exist a node Γ-cover $\mathcal{U}$ of G with $\lvert\mathcal{U}\rvert \leq k$?

Definition 38. A family of sets $\mathcal{S} \subseteq \mathcal{P}(V)$ is called a *sigma* Γ*-cover* if the following conditions hold:

1. for each $S \in \mathcal{S}$, the induced subgraph $G[S]$ belongs to the class Γ
2. for each edge $e \in E$, there exists a set $S \in \mathcal{S}$ such that $e \in E(G[S])$.

The *weight* of a sigma Γ-cover $\mathcal{S}$, denoted by $\texttt{wgt}(\mathcal{S})$, is defined as

$$\texttt{wgt}(\mathcal{S}) := \sum_{S \in \mathcal{S}} |S|$$

.

The corresponding decision problem is defined as follows:

Sigma Γ-Cover
Input: An undirected graph $G = (V, E)$, and a positive integer k.
Question: Does there exist a sigma Γ-cover $\mathcal{S}$ of G such that $\texttt{wgt}(\mathcal{S}) \leq k$?

Definition 39. *For a graph G, a newly added vertex is called a universal vertex if it is adjacent to every vertex in G.*

We are allowed to add more than one universal vertex. Given a graph G, if r universal vertices are added, then the resulting graph, denoted by G^r, can be defined as follows:

$$G^r := (V(G) \cup \{u_1, \ldots, u_r\}, E(G) \cup \{u_i v \mid 1 \leq i \leq r, v \in V\}).$$

In words, the new graph G^r consists of the original graph G together with r new universal vertices, each adjacent to every vertex in G. We now have sufficient notions to prove the second part of the conjecture.

Theorem 118. *If $K_2 \in \Gamma$, and Γ is closed under the deletion and addition of universal vertices, then* Node Γ-Cover $\leq_{\text{p}}$ Sigma Γ-Cover.

Proof. Let (G, k) be an instance of Node Γ-Cover. We construct an instance (G', k') of Sigma Γ-Cover as follows:

$$G' := (V(G) \cup \{u_1, u_2, \ldots, u_{2m+1}\}, E(G) \cup \{u_i v \mid 1 \leq i \leq 2m+1, v \in V(G)\})$$

and

$$k' = n + k(2m+1) + 2m.$$

We now show that (G, k) and (G', k') are equivalent instances. For the forward direction, suppose $\mathcal{U}$ is a node Γ-cover of G with $|\mathcal{U}| \leq k$. Define

$$\mathcal{A} := \{U \cup \{u_1, u_2, \ldots, u_{2m+1}\} \mid U \in \mathcal{U}\}$$

$$\mathcal{B} := \{\{v_1, v_2\} : v_1 v_2 \in E(G)\}$$

We claim that $\mathcal{A} \cup \mathcal{B}$ is a sigma Γ-cover of G' with $\texttt{wgt}(\mathcal{A} \cup \mathcal{B}) \leq n + k(2m+1) + 2m$. Each element of $\mathcal{B}$ is a K_2 and by assumption $K_2 \in \Gamma$. Furthermore, since $U \in \Gamma$ and Γ is closed under the addition of universal vertices, it follows that $U \cup \{u_1, u_2, \ldots, u_{2m+1}\} \in \Gamma$ for every $U \in \mathcal{U}$. It is clear that $\mathcal{A} \cup \mathcal{B}$ covers all edges of G'. We now compute the weight:

$$\begin{aligned}
\texttt{wgt}(\mathcal{A}) &= \sum_{U \in \mathcal{U}} |U \cup \{u_1, u_2, \ldots, u_{2m+1}\}| \\
&= \sum_{U \in \mathcal{U}} (|U| + 2m + 1) \\
&\leq \sum_{U \in \mathcal{U}} |U| + k(2m+1) \\
&= n + k(2m+1)
\end{aligned}$$

and $\texttt{wgt}(\mathcal{B}) = 2m$. Therefore, $\texttt{wgt}(\mathcal{A} \cup \mathcal{B}) = \texttt{wgt}(\mathcal{A}) + \texttt{wgt}(\mathcal{B}) \leq n + k(2m+1) + 2m$.

For the reverse direction, suppose $\mathcal{S}$ is a sigma Γ-cover of G' with $\texttt{wgt}(\mathcal{S}) \leq n + k(2m+1) + 2m$. We claim that there exists a vertex $u^* \in \{u_1, u_2, \ldots, u_{2m+1}\}$ that belongs to at most k elements of $\mathcal{S}$. For the sake of contraction, assume that each u_i belongs to at least $k+1$ elements of $\mathcal{S}$. Then we have

$$
\begin{aligned}
\mathtt{wgt}(\mathcal{S}) &= \sum_{v \in V(G')} val_{\mathcal{S}}(v) \\
&= \sum_{v \in V(G)} val_{\mathcal{S}}(v) + \sum_{v \in \{u_1, u_2, \ldots, u_{2m+1}\}} val_{\mathcal{S}}(v) \\
&\geq \sum_{v \in V(G)} 1 + \sum_{v \in \{u_1, u_2, \ldots, u_{2m+1}\}} (k+1) \\
&\geq n + (k+1)(2m+1) \\
&= n + k(2m+1) + 2m + 1 \\
&\geq \mathtt{wgt}(\mathcal{S}) + 1
\end{aligned}
$$

which is a contradiction. Hence, there exists some $u^* \in \{u_1, u_2, \ldots, u_{2m+1}\}$ that belongs to at most k elements of $\mathcal{S}$.

Now, define $\mathcal{N} = \{S \setminus \{u_1, u_2, \ldots, u_{2m+1}\} \mid u^* \in S \in \mathcal{S}\}$. We claim that $\mathcal{N}$ is a node Γ-cover of G of size at most k.

For each $S \in \mathcal{S}$, we have $S \setminus \{u_1, u_2, \ldots, u_{2m+1}\} \in \Gamma$ because Γ is closed under deletion of universal vertices. This implies that all elements of $\mathcal{N}$ are in Γ.

Let $v \in V(G)$. Since $\mathcal{S}$ is sigma Γ-cover of G' and $vu^* \in E(G')$, there is some S in $\mathcal{S}$ such that $\{u^*, v\} \subseteq S$. This implies that $v \in S \setminus \{u_1, u_2, \ldots, u_{2m+1}\} \in \mathcal{N}$. Hence all vertices of G are covered by $\mathcal{N}$. Moreover, we have already established that $|\mathcal{N}| \leq k$. Therefore, we conclude that $\mathcal{N}$ is a node Γ-cover of size at most k. □

Let H be a fixed graph, then we define

$$\text{Free}_{\simeq}(H) := \{G \mid \text{no induced subgraph of } G \text{ is isomorphic to } H\}.$$

In other words, $\text{Free}_{\simeq}(H)$ is the set of all graphs G such that H is not an induced subgraph of G (under isomorphism).

Node Γ-Cover has been shown to be NP-complete [6, 26] for $\Gamma = \text{Free}_{\simeq}(H)$ where $|V(H)| > 2$. By applying Theorem 118 and NP-completeness of Node Γ-Cover for $\Gamma = \text{Free}_{\simeq}(H)$ with $|V(H)| > 2$, we can conclude that Sigma Γ-Cover is NP-hard for the same class Γ. It is easy to observe that Sigma Γ-Cover is in NP. Therefore, we obtain the

following corollary.

Corollary 7.6.1. *The* Sigma Γ-Cover *problem is NP-complete for* $\Gamma = Free_{\simeq}(H)$ *such that* $K_2 \in \Gamma, \Gamma$ *is closed under deletion and addition of universal vertices and* $|V(H)| > 2$.

7.7 Conclusions

In this chapter, we investigated the classical complexity of the $\mathcal{F}$-Vertex Splitting problem for various graph classes $\mathcal{F}$, under both inclusive and exclusive variants of vertex splitting. These two variants capture fundamentally different structural requirements, and our results provide a nuanced understanding of their algorithmic behavior.

We showed that the Cycle Graph-Exclusive Vertex Splitting problem admits a polynomial-time algorithm, identifying a notable tractable case. However, the complexity of the corresponding inclusive variant remains unresolved. On the intractability side, we established several NP-completeness results. In particular, we proved that Star Forest-Inclusive Vertex Splitting is NP-complete, even when restricted to planar graphs of maximum degree three. This hardness result extends to more structured graph classes, such as Uniform Cycle Graphs and Uniform Bicluster Graphs. Furthermore, we demonstrated that both Bipartite-Inclusive Vertex Splitting and Bipartite-Exclusive Vertex Splitting are computationally equivalent to the well-known Odd Cycle Transversal problem.

Finally, we addressed a conjecture posed in [58], and established the second part by proving that under suitable closure properties of the class Γ, the Node Γ-Cover problem polynomially reduces to the Sigma Γ-Cover problem. The first part of the conjecture, however, remains open and presents an intriguing direction for further research.

Chapter 8

Conclusions and Open Problems

In this thesis, we have conducted an in-depth study of the parameterized complexity of several graph-theoretic problems. The problems investigated include Uniform Cluster Graph Modifications, MaxMin Degree-d-Modulator, MaxMin Feedback Vertex Set, MaxMin Multiway Cut-Uncut, MaxMin Separator-Unseparator, MaxMin Subset Feedback Vertex Set, MaxMin List Allocation, and $\mathcal{F}$-Vertex Splitting. For the majority of these problems, the primary parameter considered is the solution size. However, in certain cases, we have also explored the parameterized complexity with respect to structural graph parameters such as vertex cover number, feedback vertex set number, and treewidth. Our contributions include both fixed-parameter tractable (FPT) algorithms and hardness results, thereby enriching the parameterized complexity landscape of the studied problems.

In Chapter 3, we addressed several open problems posed by Misra, Mittal, Saurabh, and Thakkar [95] concerning the complexity of the Uniform Cluster Graph Modification problem. Our contributions not only resolve these questions but also yield improvements in both time complexity and kernel size over existing results, including those in [92, 95, 106].

Chapter 4 focuses on the relatively unexplored MaxMin Degree-d-Modulator problem, which can be viewed as a natural generalization of the work by Zehavi [114]. We provide the first fixed-parameter tractability and kernelization results for this problem.

In Chapter 5, we examined the parameterized complexity of the MaxMin variant of the

classical Feedback Vertex Set problem. Our results build upon traditional approaches and provide new algorithmic insights, particularly when parameterized by the structural parameter vertex cover.

Chapter 6 extends several known separation problems to their MaxMin variants. Specifically, we introduced and analyzed MaxMin Multiway Cut-Uncut, which generalizes results by Duarte et al. [47], and MaxMin Separator-Unseparator, which extends the work of Gaikwad, Kumar, Maity, Saurabh, and Sharma [66]. Additionally, we studied the MaxMin List Allocation problem, a generalization of the well-known Multiway Cut problem, originally introduced by Kim, Paul, Sau, and Thilikos [85].

Finally, in Chapter 7, we investigated the classical complexity of the $\mathcal{F}$-Vertex Splitting problem, which has recently attracted significant attention in the literature [4, 5, 9, 10, 14, 59, 61, 70]. Our analysis distinguishes between inclusive and exclusive variants and provides a fine-grained complexity classification for several important graph classes.

While this thesis advances the understanding of several graph modification and separation problems through the lens of parameterized complexity, many compelling questions remain open. We list below some of the key open problems that arise directly from our work, as well as a few promising directions for future research:

- Does there exist a linear or quadratic vertex kernel for Uniform Cluster Vertex Deletion (UCVD)?
- Can we design a $(2-\epsilon)^k \cdot n^{\mathcal{O}(1)}$-time algorithm for UCVD?
- Are there $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$-time algorithms for Uniform Cluster Inclusive Vertex Splitting (UCIVS) and Uniform Cluster Exclusive Vertex Splitting (UCEVS)?
- Does Uniform Cluster Edge Editing (UCEE) admit a linear vertex kernel?
- For $d \in \{1, 2\}$, can the constant c in the $\mathcal{O}^*(c^k)$-time algorithm be significantly reduced?
- Can we get an FPT algorithm for MaxMin Degree-d-Modulator with running time $\mathcal{O}^*(c^k)$ for $d \geq 3$?

- Can FPT algorithms be designed for MaxMin Multiway Cut-Uncut, MaxMin Separator-Unseparator, and MaxMin List Allocation under restricted settings?
- What is the classical complexity of the inclusive variant of Cycle Graph-Vertex Splitting?
- Can structural parameters like *modular-width*, *neighborhood diversity*, or *clique-width* lead to improved FPT algorithms or kernelization results for the studied problems?

List of Publications and Manuscripts arising from the thesis

1. **Parameterized Algorithms for Editing to Uniform Cluster Graph** | with Ajinkya Gaikwad, Soumen Maity. In 25th International Symposium on Fundamentals of Computation Theory **(FCT 2025)** Article No.13; pp.165-179

2. **Parameterized Algorithms for Editing to Uniform Cluster Graph** | with Ajinkya Gaikwad, Soumen Maity.(Full Version of the paper accepted in FCT 2025: Submitted to a Journal)

3. **Maximum Minimal Feedback Vertex Set: A Parameterized Perspective** | with Ajinkya Gaikwad, Soumen Maity, Saket Saurabh, Shuvam Kant Tripathi.(Uploaded on ArXiv)

4. **Inclusive and Exclusive Vertex Splitting into Specific Graph Classes: NP Hardness and Algorithms**| with Ajinkya Gaikwad, S. Padmapriya, Praneet Kumar Patra, Harsh Sanklecha and Soumen Maity.(Uploaded on ArXiv)

5. **When Optimization Equals Feasibility: The Case of MaxMin Separation with Uncut Constraints** | with Ajinkya Gaikwad, Soumen Maity, Saket Saurabh, Roohani Sharma. (Submitted to a Conference)

Bibliography

[1] Hassan Aboueisha, Shahid Hussain, Vadim Lozin, Jérôme Monnot, Bernard Ries, and Viktor Zamaraev. Upper domination: Towards a dichotomy through boundary properties. *Algorithmica*, 80(10):2799–2817, 2018. doi:10.1007/s00453-017-0346-9. 1.3, 1.3

[2] Faisal N. Abu-Khzam. A kernelization algorithm for d-hitting set. *Journal of Computer and System Sciences*, 76(7):524–531, 2010. doi:10.1016/j.jcss.2009.09.002. 1.2

[3] Faisal N. Abu-Khzam, Emmanuel Arrighi, Matthias Bentert, Pål Grønås Drange, Judith Egan, Serge Gaspers, Alexis Shaw, Peter Shaw, Blair D. Sullivan, and Petra Wolf. Cluster editing with vertex splitting. *Discrete Applied Mathematics*, 371:185–195, 2025. doi:10.1016/j.dam.2025.04.013. 1.2, 2.5, 7

[4] Faisal N. Abu-Khzam, Tom Davot, Lucas Isenmann, and Sergio Thoumi. On the complexity of 2-club cluster editing with vertex splitting, 2024. arXiv:2411.04846. 1.5, 8

[5] Faisal N. Abu-Khzam and Sergio Thoumi. On the complexity of claw-free vertex splitting, 2025. arXiv:2506.06044. 1.5, 8

[6] Demetrios Achlioptas. The complexity of g-free colourability. *Discrete Mathematics*, 165-166:21–30, 1997. doi:10.1016/S0012-365X(97)84217-3. 7.6

[7] Júlio Araújo, Marin Bougeret, Victor Campos, and Ignasi Sau. Introducing lop-kernels: A framework for kernelization lower bounds. *Algorithmica*, 84(11):3365–3406, 2022. doi:10.1007/s00453-022-00979-z. 1.5, 5

[8] Sanjeev Arora, David R. Karger, and Marek Karpinski. Polynomial time approximation schemes for dense instances of np-hard problems. *Journal of Computer and System Sciences*, 58(1):193–210, 1999. doi:10.1006/JCSS.1998.1605. 29

[9] Emmanuel Arrighi, Matthias Bentert, Pål Grønås Drange, Blair D. Sullivan, and Petra Wolf. Cluster Editing with Overlapping Communities. In *IPEC 2023*, volume 285 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 2:1–2:12, 2023. doi:10.4230/LIPIcs.IPEC.2023.2. 1.4, 8

[10] Jakob Baumann, Matthias Pfretzschner, and Ignaz Rutter. Parameterized complexity of vertex splitting to pathwidth at most 1. *Theoretical Computer Science*, 1021:114928, 2024. doi:10.1016/j.tcs.2024.114928. 1.5, 8

[11] Cristina Bazgan, Ljiljana Brankovic, Katrin Casel, Henning Fernau, Klaus Jansen, Kim-Manuel Klein, Michael Lampis, Mathieu Liedloff, Jérôme Monnot, and Vangelis Th. Paschos. The many facets of upper domination. *Theoretical Computer Science*, 717:2–25, 2018. doi:10.1016/j.tcs.2017.05.042. 1.3

[12] Rémy Belmonte, Eun Jung Kim, Michael Lampis, Valia Mitsou, and Yota Otachi. Grundy distinguishes treewidth from pathwidth. *SIAM Journal on Discrete Mathematics*, 36(3):1761–1787, 2022. doi:10.1137/20M1385779. 1.3

[13] Amir Ben-Dor, Ron Shamir, and Zohar Yakhini. Clustering gene expression patterns. *Journal of Computational Biology*, 6(3-4):281–297, 1999. doi:10.1089/106652799318274. 1.2

[14] Matthias Bentert, Alex Crane, Pål Grønås Drange, Felix Reidl, and Blair D. Sullivan. Correlation Clustering with Vertex Splitting. In *SWAT 2024*, volume 294 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:17, 2024. doi:10.4230/LIPIcs.SWAT.2024.8. 1.5, 2.5, 7.5, 8

[15] Nadja Betzler, Robert Bredereck, Rolf Niedermeier, and Johannes Uhlmann. On bounded-degree vertex deletion parameterized by treewidth. *Discrete Applied Mathematics*, 160(1):53–60, 2012. doi:10.1016/j.dam.2011.08.013. 1.3, 1.5, 4

[16] Dan Bienstock. On the complexity of testing for odd holes and induced odd paths. *Discrete Mathematics*, 90(1):85–92, 1991. doi:10.1016/0012-365X(91)90098-M. 1.5, 6.2

[17] Mostafa Blidia, Mustapha Chellali, Odile Favaron, and Nacéra Meddah. Maximal k-independent sets in graphs. *Discussiones Mathematicae Graph Theory*, 28(1):151–163, 2008. URL: http://eudml.org/doc/270197. 1.5

[18] Sebastian Böcker, Sebastian Briesemeister, Quang Bao Anh Bui, and Anke Truß. A fixed-parameter approach for weighted cluster editing. In *APBC 2008*, volume 6 of *Advances in Bioinformatics and Computational Biology*, pages 211–220, 2008. URL: http://www.comp.nus.edu.sg/%7Ewongls/psZ/apbc2008/apbc050a.pdf. 1.2

[19] Sebastian Böcker, Sebastian Briesemeister, and Gunnar W. Klau. Exact algorithms for cluster editing: Evaluation and experiments. In *Experimental Algorithms*, pages 289–302, 2008. 1.2

[20] Sebastian Böcker and Peter Damaschke. Even faster parameterized cluster deletion and cluster editing. *Information Processing Letters*, 111(14):717–721, 2011. doi:10.1016/j.ipl.2011.05.003. 1.2, 3.4.3, 3.4.3

[21] Hans L. Bodlaender, Michael R. Fellows, Pinar Heggernes, Federico Mancini, Charis Papadopoulos, and Frances Rosamond. Clustering with partial information. *Theoretical Computer Science*, 411(7):1202–1211, 2010. doi:10.1016/j.tcs.2009.12.016. 1.2

[22] Hans L. Bodlander. On disjoint cycles. *International Journal of Foundations of Computer Science*, 05(01):59–68, 1994. doi:10.1142/S0129054194000049. 5

[23] Anudhyan Boral, Marek Cygan, Tomasz Kociumaka, and Marcin Pilipczuk. A fast branching algorithm for cluster vertex deletion. *Theory of Computing Systems*, 58(2):357–376, 2016. doi:10.1007/s00224-015-9631-7. 1.2

[24] Nicolas Boria, Federico Della Croce, and Vangelis Th. Paschos. On the max min vertex cover problem. *Discrete Applied Mathematics*, 196:62–71, 2015. doi:10.1016/j.dam.2014.06.001. 1.3, 1.3, 5

[25] Arman Boyacı and Jérôme Monnot. Weighted upper domination number. *Electronic Notes in Discrete Mathematics*, 62:171–176, 2017. doi:10.1016/j.endm.2017.10.030. 1.3

[26] Jason I. Brown. The complexity of generalized graph colorings. *Discrete Applied Mathematics*, 69(3):257–270, 1996. doi:10.1016/0166-218X(96)00096-0. 7.6

[27] Pablo Burzyn, Flavia Bonomo, and Guillermo Duran. NP-completeness results for edge modification problems. *Discrete Applied Mathematics*, 154(13):1824–1844, 2006. doi:10.1016/j.dam.2006.03.031. 1.1

[28] Sebastian Böcker. A golden ratio parameterized algorithm for cluster editing. *Journal of Discrete Algorithms*, 16:79–89, 2012. doi:10.1016/j.jda.2012.04.005. 1.2

[29] Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996. doi:10.1016/0020-0190(96)00050-6. 1.1, 1.2

[30] Yixin Cao and Jianer Chen. Cluster editing: Kernelization based on edge cuts. In *IPEC 2010*, volume 6478 of *Lecture Notes in Computer Science*, pages 60–71, 2010. doi:10.1007/978-3-642-17493-3_8. 1.2

[31] Yixin Cao and Yuping Ke. Improved Kernels for Edge Modification Problems. In *IPEC 2021*, volume 214 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:14, 2021. doi:10.4230/LIPIcs.IPEC.2021.13. 1.2

[32] Karthekeyan Chandrasekaran and Weihang Wang. Fixed parameter approximation scheme for min-max k-cut. *Mathematical Programming*, 197(2):1093–1144, 2023. 1.3, 1.5

[33] Juhi Chaudhary, Sounaka Mishra, and B.S. Panda. Minimum maximal acyclic matching in proper interval graphs. *Discrete Applied Mathematics*, 360:414–427, 2025. doi:10.1016/j.dam.2024.10.012. 1.3

[34] Grant A. Cheston, G. Fricke, S.T. Hedetniemi, and D. Pokrass Jacobs. On the computational complexity of upper fractional domination. *Discrete Applied Mathematics*, 27(3):195–207, 1990. doi:10.1016/0166-218X(90)90065-K. 1.3

[35] Stephen A. Cook. The complexity of theorem-proving procedures. In *STOC*, page 151–158, 1971. doi:10.1145/800157.805047. 1

[36] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction To Algorithms*. Mit Electrical Engineering and Computer Science. MIT Press, 2001. URL: https://books.google.co.in/books?id=NLngYyWFl_YC. 2.2, 6.3

[37] B. Courcelle, J. A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33(2):125–150, 2000. 1.3

[38] Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H. 1.3, 1.5, 4

[39] Christophe Crespelle, Pål Grønås Drange, Fedor V. Fomin, and Petr Golovach. A survey of parameterized algorithms and the complexity of edge modification. *Computer Science Review*, 48:100556, 2023. doi:10.1016/j.cosrev.2023.100556. 1.1

[40] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3. 1.1, 2.2, 1, 2.2, 2, 3, 2.2, 2.3, 2.4, 5.1, 6, 75, 6.2.1

[41] Peter Damaschke. Bounded-degree techniques accelerate some parameterized graph algorithms. In *IWPEC 2009*, volume 5917 of *Lecture Notes in Computer Science*, pages 98–109. Springer, 2009. doi:10.1007/978-3-642-11269-0_8. 1.2, 3.4.3, 53

[42] Peter Damaschke. Fixed-parameter enumerability of cluster editing and related problems. *Theory of Computing Systems*, 46(2):261–283, 2010. doi:10.1007/s00224-008-9130-1. 1.2

[43] Michael Doob. On characterizing certain graphs with four eigenvalues by their spectra. *Linear Algebra and its Applications*, 3(4):461–482, 1970. doi:10.1016/0024-3795(70)90037-6. 1.5, 2.1

[44] Dorit Dor and Michael Tarsi. Graph decomposition is NPC - a complete proof of holyer's conjecture. In *STOC*, page 252–263, 1992. doi:10.1145/129712.129737. 7.5

[45] Rod G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, 24:873–921, 1995. 5

[46] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer Publishing Company, Incorporated, 2012. 2.2

[47] Gabriel L Duarte, Hiroshi Eto, Tesshu Hanaka, Yasuaki Kobayashi, Yusuke Kobayashi, Daniel Lokshtanov, Lehilton LC Pedrosa, Rafael CS Schouery, and Uéverton S Souza. Computing the largest bond and the maximum connected cut of a graph. *Algorithmica*, 83:1421–1458, 2021. 1.5, 6, 6.1, 6.3, 93, 6.3, 8

[48] Louis Dublois, Tesshu Hanaka, Mehdi Khosravian Ghadikolaei, Michael Lampis, and Nikolaos Melissinos. (in)approximability of maximum minimal fvs. *Journal of Computer and System Sciences*, 124:26–40, 2022. doi:10.1016/j.jcss.2021.09.001. 1.3, 1.5, 5, 6

[49] Paul Erdös and Richard Rado. Intersection theorems for systems of sets. *Journal of the London Mathematical Society*, 1(1):85–90, 1960. 75

[50] Hiroshi Eto, Tesshu Hanaka, Yasuaki Kobayashi, and Yusuke Kobayashi. Parameterized Algorithms for Maximum Cut with Connectivity Constraints. In *IPEC 2019*, volume 148 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:15, 2019. doi:10.4230/LIPIcs.IPEC.2019.13. 1.3

[51] Leonhard Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, pages 128–140, 1741. 107

[52] L. Faria, C.M.H. de Figueiredo, and C.F.X. Mendonca. splitting number is np-complete. *Discrete Applied Mathematics*, 108(1):65–83, 2001. doi:10.1016/S0166-218X(00)00220-1. 1.5

[53] Michael R Fellows. The robertson-seymour theorems: A survey of applications. *Contemporary Mathematics*, 89:1–18, 1989. 1.5, 6.2

[54] Michael R. Fellows, Jiong Guo, Christian Komusiewicz, Rolf Niedermeier, and Johannes Uhlmann. Graph-based data clustering with overlaps. *Discrete Optimization*, 8(1):2–17, 2011. doi:10.1016/j.disopt.2010.09.006. 1.2

[55] Michael R. Fellows, Jiong Guo, Hannes Moser, and Rolf Niedermeier. A generalization of nemhauser and trotters local optimization theorem. *Journal of Computer and System Sciences*, 77(6):1141–1158, 2011. doi:10.1016/j.jcss.2010.12.001. 1.3, 1.5, 4

[56] Michael R. Fellows, Michael A. Langston, Frances A. Rosamond, and Peter Shaw. Efficient parameterized preprocessing for cluster editing. In *FCT 2007*, volume 4639 of *Lecture Notes in Computer Science*, pages 312–321, 2007. doi:10.1007/978-3-540-74240-1_27. 1.2

[57] Henning Fernau. *Parameterized algorithmics: A graph-theoretic approach*. PhD thesis, 2005. 1.5, 4

[58] Alexander Firbas. *Establishing hereditary graph properties via vertex splitting*. PhD thesis, Wien, 2023. 1.5, 1.6, 7.3, 7.6, 7.6, 7.7

[59] Alexander Firbas, Alexander Dobler, Fabian Holzer, Jakob Schafellner, Manuel Sorge, Anaïs Villedieu, and Monika Wißmann. The complexity of cluster vertex splitting and company. *Discrete Applied Mathematics*, 365:190–207, 2025. doi:10.1016/j.dam.2025.01.012. 1.2, 1.5, 7.5, 7.6, 8

[60] Alexander Firbas and Manuel Sorge. On the Complexity of Establishing Hereditary Graph Properties via Vertex Splitting. In *ISAAC 2024*, volume 322 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:15, 2024. doi:10.4230/LIPIcs.ISAAC.2024.30. 1.5, 7.3

[61] Alexander Firbas and Manuel Sorge. On the complexity of establishing hereditary graph properties via vertex splitting. 2024. URL: https://arxiv.org/abs/2401.16296. 3.2.2, 25, 26, 3.2.2, 3.2.2, 3.2.2, 3.2.2, 8

[62] Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: theory of parameterized preprocessing*. Cambridge University Press, 2019. 1.1

[63] Fabio Furini, Ivana Ljubić, and Markus Sinnl. An effective dynamic programming algorithm for the minimum-cost maximal knapsack packing problem. *European Journal of Operational Research*, 262(2):438–448, 2017. doi:10.1016/j.ejor.2017.03.061. 1.3

[64] Ajinkya Gaikwad, Hitendra Kumar, and Soumen Maity. Parameterized algorithms for editing to uniform cluster graph, 2025. arXiv:2404.10023. 1.6, 3

[65] Ajinkya Gaikwad, Hitendra Kumar, and Soumen Maity. Parameterized algorithms for editing to uniform cluster graph. In *FCT 2025*, page 165–179, 2025. doi:10.1007/978-3-032-04700-7_13. 1.6, 3

[66] Ajinkya Gaikwad, Hitendra Kumar, Soumen Maity, Saket Saurabh, and Roohani Sharma. MaxMin Separation Problems: FPT Algorithms for st-Separator and Odd Cycle Transversal. In *STACS 2025*, volume 327 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 36:1–36:21, 2025. doi:10.4230/LIPIcs.STACS.2025.36. 1.3, 1.5, 6, 8

[67] Ajinkya Gaikwad, Hitendra Kumar, Soumen Maity, Saket Saurabh, and Shuvam Kant Tripathi. Maximum minimal feedback vertex set: A parameterized perspective, 2022. arXiv:2208.01953. 1.6, 5

[68] Ajinkya Gaikwad, Hitendra Kumar, S. Padmapriya, Praneet Kumar Patra, Harsh Sanklecha, and Soumen Maity. Inclusive and exclusive vertex splitting into specific graph classes: NP hardness and algorithms, 2025. URL: https://arxiv.org/abs/2510.26938, arXiv:2510.26938. 1.6

[69] Ajinkya Gaikwad and Soumen Maity. Parameterized complexity of upper edge domination, 2022. arXiv:2208.02522. 1.3

[70] Felix Goldberg, Steve Kirkland, Anu Varghese, and Ambat Vijayakumar. On split graphs with four distinct eigenvalues. *Discrete Applied Mathematics*, 277:163–171, 2020. doi:10.1016/j.dam.2019.09.016. 1.5, 2.1, 8

[71] Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Automated generation of search tree algorithms for hard graph modification problems. *Algorithmica*, 39(4):321–347, 2004. doi:10.1007/s00453-004-1090-5. 1.2

[72] Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Graph-modeled data clustering: Exact algorithms for clique generation. *Theory of Computing Systems*, 38:373–392, 2005. 1.2

[73] Jiong Guo. A more effective linear kernelization for cluster editing. *Theoretical Computer Science*, 410(8):718–726, 2009. doi:10.1016/j.tcs.2008.10.021. 1.2

[74] Jiong Guo, Iyad A. Kanj, Christian Komusiewicz, and Johannes Uhlmann. Editing graphs into disjoint unions of dense clusters. *Algorithmica*, 61(4):949–970, 2011. doi:10.1007/S00453-011-9487-4. 1.2

[75] Jiong Guo, Christian Komusiewicz, Rolf Niedermeier, and Johannes Uhlmann. A more relaxed model for graph-based data clustering: s-plex cluster editing. *SIAM Journal on Discrete Mathematics*, 24(4):1662–1683, 2010. doi:10.1137/090767285. 1.2

[76] Tesshu Hanaka, Hans L. Bodlaender, Tom C. van der Zanden, and Hirotaka Ono. On the maximum weight minimal separator. *Theoretical Computer Science*, 796:294–308, 2019. doi:10.1016/j.tcs.2019.09.025. 1.3, 1.3

[77] Michael A. Henning and D. Pradhan. Algorithmic aspects of upper paired-domination in graphs. *Theoretical Computer Science*, 804:98–114, 2020. doi:10.1016/j.tcs.2019.10.045. 1.3

[78] Ian Holyer. The NP-completeness of some edge-partition problems. *SIAM Journal on Computing*, 10(4):713–717, 1981. doi:10.1137/0210054. 3.2, 3.2, 7.4

[79] Falk Hüffner, Christian Komusiewicz, Hannes Moser, and Rolf Niedermeier. Fixed-parameter algorithms for cluster vertex deletion. In *LATIN 2008: Theoretical Informatics*, pages 711–722, 2008. 1.2

[80] Yoichi Iwata and Yusuke Kobayashi. Improved Analysis of Highest-Degree Branching for Feedback Vertex Set. In *IPEC 2019*, volume 148 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22:1–22:11, 2019. doi:10.4230/LIPIcs.IPEC.2019.22. 5

[81] Michael S. Jacobson and Ken Peters. Chordal graphs and upper irredundance, upper domination and independence. In *Topics on Domination*, volume 48 of *Annals of Discrete Mathematics*, pages 59–69. 1991. doi:10.1016/S0167-5060(08)71038-0. 1.3, 1.3

[82] Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, 1972. doi:10.1007/978-1-4684-2001-2_9. 1

[83] Richard M. Karp. On the computational complexity of combinatorial problems. *Networks*, 5(4):45–68, 1975. doi:10.1002/NET.1975.5.1.45. 1.5, 6.2

[84] Marek Karpinski and Warren Schudy. Linear time approximation schemes for the gale-berlekamp game and related minimization problems. In *STOC*, page 313–322, 2009. doi:10.1145/1536414.1536458. 29

[85] Eun Jung Kim, Christophe Paul, Ignasi Sau, and Dimitrios M. Thilikos. Parameterized algorithms for min-max multiway cut and list digraph homomorphism. *Journal of Computer and System Sciences*, 86:191–206, 2017. doi:10.1016/j.jcss.2017.01.003. 1.3, 1.5, 6, 8

[86] Christian Komusiewicz and Johannes Uhlmann. Alternative parameterizations for cluster editing. In *SOFSEM 2011: Theory and Practice of Computer Science*, pages 344–355, 2011. 1.2

[87] Michael Lampis, Nikolaos Melissinos, and Manolis Vasilakis. Parameterized Max Min Feedback Vertex Set. In *MFCS 2023*, volume 272 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 62:1–62:15, 2023. doi:10.4230/LIPIcs.MFCS.2023.62. 1.3, 1.5, 5.3

[88] John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4. 1.1

[89] Jason Li and Jesper Nederlof. Detecting feedback vertex sets of size k in $O^*(2.7^k)$ time. In *SODA 2020*, pages 971–989. SIAM, 2020. doi:10.1137/1.9781611975994.58. 5

[90] William Lochet, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Exploiting Dense Structures in Parameterized Complexity. In *STACS 2021*, volume 187 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 50:1–50:17, 2021. doi:10.4230/LIPIcs.STACS.2021.50. 56

[91] Daniel Lokshtanov, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Reducing CMSO model checking to highly connected graphs. In *ICALP 2018*, volume 107 of *LIPIcs*, pages 135:1–135:14, 2018. doi:10.4230/LIPICS.ICALP.2018.135. 2.4, 4, 6, 74, 6, 6.1

[92] Jayakrishnan Madathil and Kitty Meeks. Parameterized algorithms for balanced cluster edge modification problems, 2024. arXiv:2403.03830. 1.5, 8

[93] Alice Anne McRae. *Generalizing NP-completeness proofs for bipartite graphs and chordal graphs*. Clemson University, 1994. 1.3

[94] Mishra, Sounaka and Sikdar, Kripasindhu. On the hardness of approximating some np-optimization problems related to minimum linear ordering problem. *RAIRO-Theor. Inf. Appl.*, 35(3):287–309, 2001. doi:10.1051/ita:2001121. 1.3, 1.5, 5

[95] Neeldhara Misra, Harshil Mittal, Saket Saurabh, and Dhara Thakkar. On the Complexity of the Eigenvalue Deletion Problem. In *ISAAC 2023*, volume 283 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 53:1–53:17, 2023. doi:10.4230/LIPIcs.ISAAC.2023.53. 1.5, 8

[96] Jérôme Monnot, Henning Fernau, and David Manlove. Algorithmic aspects of upper edge domination. *Theoretical Computer Science*, 877:46–57, 2021. doi:10.1016/j.tcs.2021.03.038. 1.3

[97] Hannes Moser, Rolf Niedermeier, and Manuel Sorge. Exact combinatorial algorithms and experiments for finding maximum k-plexes. *Journal of Combinatorial Theory*, 24(3):347–373, 2012. doi:10.1007/s10878-011-9391-5. 1.3, 1.5, 4

[98] Assaf Natanzon, Ron Shamir, and Roded Sharan. Complexity classification of some edge modification problems. *Discrete Applied Mathematics*, 113(1):109–128, 2001. doi:10.1016/S0166-218X(00)00391-7. 1.1

[99] Naomi Nishimura, Prabhakar Ragde, and Dimitrios M. Thilikos. Fast fixed-parameter tractable algorithms for nontrivial generalizations of vertex cover. *Discrete Applied Mathematics*, 152(1):229–245, 2005. doi:10.1016/j.dam.2005.02.029. 1.3, 1.5, 4

[100] Martin Nöllenburg, Manuel Sorge, Soeren Terziadis, Anaïs Villedieu, Hsiang-Yun Wu, and Jules Wulms. Planarizing graphs and their drawings by vertex splitting. *Journal of Computational Geometry*, 16(1):333–372, 2025. doi:10.20382/jocg.v16i1a10. 1.5

[101] Julius Petersen. Die theorie der regulären graphs. *Acta Mathematica*, 15:193–220, 1891. URL: https://api.semanticscholar.org/CorpusID:123779343. 107

[102] Fábio Protti, Maise Dantas da Silva, and Jayme Luiz Szwarcfiter. Applying modular decomposition to parameterized cluster editing problems. *Theory of Computing Systems*, 44(1):91–104, 2008. doi:10.1007/s00224-007-9032-7. 1.2

[103] Neil Robertson and Paul D Seymour. Graph minors. III. planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984. doi:10.1016/0095-8956(84)90013-3. 2.3

[104] Neil Robertson and Paul D. Seymour. Graph minors. IX. disjoint crossed paths. *Journal of Combinatorial Theory B*, 49(1):40–77, 1990. doi:10.1016/0095-8956(90)90063-6. 1.5, 6

[105] Neil Robertson and Paul D Seymour. Graph minors. XIII. the disjoint paths problem. *Journal of combinatorial theory, Series B*, 63(1):65–110, 1995. 1.5, 6, 82

[106] Andreas Steinvik. Kernelization for balanced graph clustering. Master's thesis, The University of Bergen, 2020. 1.5, 8

[107] Prafullkumar Prabhakar Tale. Some results on graph contraction problems. 2020. 1.1

[108] Dekel Tsur. Faster parameterized algorithm for cluster vertex deletion. *Theory of Computing Systems*, 65:1–21, 2021. doi:10.1007/s00224-020-10005-w. 1.2, 3.1

[109] Dekel Tsur. Cluster deletion revisited. *Information Processing Letters*, 173:106171, 2022. doi:10.1016/j.ipl.2021.106171. 1.2

[110] Pim van't Hof, Daniël Paulusma, and Gerhard J. Woeginger. Partitioning graphs into connected parts. *Theoretical Computer Science*, 410(47):4834–4843, 2009. doi:10.1016/j.tcs.2009.06.028. 1.5, 6.2

[111] Zhengren Wang, Yi Zhou, Mingyu Xiao, and Bakhadyr Khoussainov. Listing maximal k-plexes in large real-world graphs. In *Proceedings of the ACM Web Conference 2022*, page 1517–1527, 2022. doi:10.1145/3485447.3512198. 1.5

[112] Douglas B. West. *Introduction to Graph Theory*. Prentice Hall, 2 edition, 2000. 2.1

[113] Mihalis Yannakakis. Node-and edge-deletion NP-complete problems. In *STOC*, page 253–264, 1978. doi:10.1145/800133.804355. 1.5, 6

[114] Meirav Zehavi. Maximum minimal vertex cover parameterized by vertex cover. *SIAM Journal on Discrete Mathematics*, 31(4):2440–2456, 2017. doi:10.1137/16M109017X. 1.3, 1.3, 1.5, 4, 5, 8

[115] Igor E. Zvervich and Vadim E. Zverovich. An induced subgraph characterization of domination perfect graphs. *Journal of Graph Theory*, 20(3):375–395, 1995. doi:10.1002/jgt.3190200313. 1.3