

New Directions in Parameterized Deletion Problems

By

Ashwin Jacob

MATH10201604003

The Institute of Mathematical Sciences, Chennai

A thesis submitted to the

Board of Studies in Mathematical Sciences

In partial fulfillment of requirements

for the Degree of

DOCTOR OF PHILOSOPHY

of

HOMI BHABHA NATIONAL INSTITUTE

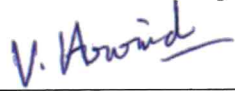


August, 2022

Homi Bhabha National Institute

Recommendations of the Viva Voce Committee

As members of the Viva Voce Committee, we certify that we have read the dissertation prepared by Ashwin Jacob entitled "New Directions in Parameterized Deletion Problems" and recommend that it may be accepted as fulfilling the thesis requirement for the award of Degree of Doctor of Philosophy.



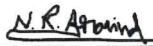
Chairman - Vikraman Arvind

Date: August 10, 2022



Guide/Convenor - Venkatesh Raman

Date: August 10, 2022



Examiner - N. R. Aravind

Date: August 10, 2022



Member 1 - Geevarghese Philip

Date: August 10, 2022



Member 2 - Saket Saurabh

Date: August 10, 2022



Member 3 - Vikram Sharma

Date: August 10, 2022

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

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

Date: August 10, 2022

Place: Chennai



Guide

STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at Homi Bhabha National Institute (HBNI) and is deposited in the Library to be made available to borrowers under rules of the HBNI.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgement of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the Competent Authority of HBNI when in his or her judgement the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.



Ashwin Jacob

DECLARATION

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

A handwritten signature in black ink, appearing to read 'Ashwin', with a long horizontal stroke underneath.

Ashwin Jacob

LIST OF PUBLICATIONS ARISING FROM THE THESIS

Journals


1. **Fixed-Parameter Tractability of $(n - k)$ List Coloring**, Aritra Banik, Ashwin Jacob, Vijay Kumar Paliwal and Venkatesh Raman, Theory of Computing Systems, 2020, Volume 64 (7) : 1307 – 1316 .
2. **Parameterized Complexity of Conflict-Free Set Cover**, Ashwin Jacob, Diptapriyo Majumdar and Venkatesh Raman, Theory of Computing Systems, 2021, Volume 65(3) : 515 – 540 .

Conferences

1. **Structural Parameterizations of Dominating Set Variants**, Dishant Goyal, Ashwin Jacob, Kaushtubh Kumar, Diptapriyo Majumdar and Venkatesh Raman, Proceedings of the 13th International Computer Science Symposium in Russia, CSR 2018, Moscow, Russia, June 6-10, 2018, pages 157-168.
2. **Deconstructing Parameterized Hardness of Fair Vertex Deletion Problems**, Ashwin Jacob, Venkatesh Raman and Vibha Sahlot, Proceedings of the 25th International Computing and Combinatorics Conference, COCOON 2019, Xi'an, China, July 29-31, 2019, pages 325-337.
3. **Structural Parameterizations with Modulator Oblivion**, Ashwin Jacob, Fahad Panolan, Venkatesh Raman and Vibha Sahlot, Proceedings of the 15th International Symposium on Parameterized and Exact Computation, IPEC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference), pages 19:1-19:18.
4. **Parameterized Complexity of Deletion to Scattered Graph Classes**, Ashwin Jacob, Diptapriyo Majumdar and Venkatesh Raman, Proceedings of the 15th International Sym-

posium on Parameterized and Exact Computation, IPEC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference), pages 18:1-18:17.

5. **Faster FPT Algorithms for Deletion to Pairs of Graph Classes**, Ashwin Jacob, Dip-tapriyo Majumdar and Venkatesh Raman, Proceedings of Fundamentals of Computation Theory: 23rd International Symposium, FCT 2021, Athens, Greece, September 12–15, 2021, pages 314-326.



Ashwin Jacob

DEDICATIONS

To my parents.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my thesis advisor, Dr. Venkatesh Raman, for his guidance, encouragement, and patience. If it hadn't been for his invaluable advice, I would have completely lost track of what to do on numerous occasions. His insightful remarks have aided me in improving my research, writing, and presentation skills. I am also grateful for the freedom he gave me to investigate various problems - sometimes for extended periods with no fruitful results. I consider myself extremely fortunate to have him as my advisor.

I thank Dr. Saket Saurabh for his encouragement and advice and for providing me with research opportunities and financial support. I would also like to thank all of the faculty of IMSc in Theoretical Computer Science for their brilliant courses, which catalyzed my evolution as a researcher.

I am deeply indebted to Dr. K. Muralikrishnan for introducing me to research during my undergraduate studies, helping me with admissions and for support throughout my research career. I thank Dr. Sunil Chandran for hosting me at IISc during my undergraduate years, helping to blossom my interests in research and considering it a career. I am grateful to Dr. Shashank K. Mehta for his advice and support during my Masters programme at IIT Kanpur.

I thank my co-authors Aritra, Dishant, Jari, Kaushtubh, Diptapriyo, Fahad and Vibha for their insightful comments and discussions. I want to thank my friends Abhranil, Gaurav, Govind, Niranka, Prateek, Pratibha, Ramit, Sanjukta, and Shivani for their contributions, discussions, listening ears, suggestions and good times.

I thank every staff member of IMSc for their help with administrative procedures, financial support, and maintaining a conducive research environment.

Finally, I would like to express my gratitude to my parents and my brother for their unwavering love, support, and understanding.

Contents

Synopsis	21
List of Figures	27
1 Introduction	31
1.1 Deletion to Scattered Graph Classes	37
1.2 Deletion distance parameterizations	39
1.3 Vertex Deletion with additional constraints	41
1.4 Organization of Thesis	43
2 Preliminaries	45
2.1 Sets, Numbers and some Notations	45
2.2 Parameterized Complexity and Kernelization	46
2.3 Graph Theory	49
2.3.1 Basic Notations and Definitions	49
2.3.2 Graph Separators	51
2.3.3 Tree Decomposition and Treewidth	53

2.4	Second Order Logic	54
2.5	Matroids	55
I	Deletion to Scattered Graph Classes	57
3	FPT algorithms for general cases	59
3.1	Deletion to scattered classes when each class is individually tractable . . .	63
3.2	Deletion to scattered classes with finite forbidden families	67
3.2.1	Iterative Compression	69
3.2.2	Finding non-separating solutions	71
3.2.3	Solving general instances	74
3.3	Conclusion	100
4	Faster Algorithms for Pairs of Scattered Graph Classes	101
4.1	Preliminaries	103
4.2	FINITE Π_1 OR Π_2 DELETION with forbidden paths	104
4.3	Π_1 OR Π_2 DELETION with a constant number of forbidden pairs	107
4.3.1	Forbidden Characterization for Π_1 OR Π_2 DELETION	107
4.3.2	The case with forbidden paths	110
4.3.3	Algorithms for Π_1 OR Π_2 DELETION without forbidden paths . .	119
4.4	Examples of SPECIAL INFINITE- (Π_1, Π_2) -DELETION	126
4.4.1	Interval or Trees	127

4.4.2	Proper Interval or Trees	135
4.4.3	Chordal or Bipartite Permutation	138
4.5	Conclusion	144
II Deletion distance parameterizations		145
5	Structural Parameterizations with Modulator Oblivion	147
5.1	Preliminaries	153
5.2	Semi Clique Tree Decomposition	153
5.3	Structural Parameterizations with Chordal Vertex Deletion Set	165
5.3.1	SETH Lower Bounds	173
5.4	Conclusion	176
6	Fixed-parameter tractability of $(n - k)$ List Coloring	177
6.1	Introduction	177
6.2	Preliminaries	179
6.3	FPT algorithm for $(n-k)$ -REGULAR LIST COLORING	180
6.4	Conclusion	187
7	Deletion Distance Parameterizations of Dominating Set Variants	189
7.1	Introduction	189
7.1.1	Motivation	189
7.1.2	Related Work	192

7.1.3	Problem Definitions	193
7.2	Dominating Set Variants parameterized by CVD Size	195
7.2.1	Upper Bounds	195
7.2.2	Lower bounds	205
7.3	Dominating Set variants parameterized by SVD size	210
7.3.1	EDS and IDS parameterized by SVD size	211
7.3.2	Improved Algorithm for EDS-SVD	212
7.3.3	Lower Bounds for IDS and EDS	217
7.4	Concluding Remarks	218

III Deletion with additional constraints 219

8 Parameterized Complexity of Conflict-Free Set Cover 221

8.1	Introduction and Previous Work	221
8.2	Hardness results for Conflict-Free Set Cover	227
8.2.1	1-Intersection Conflict-Free Set Cover parameterized by solution size k	227
8.2.2	Conflict-Free Set Cover parameterized by $ \mathcal{U} $	229
8.3	Algorithms	234
8.3.1	Conflict-Free Set Cover parameterized by solution size k	234
8.3.2	Conflict-Free Set Cover parameterized by $ \mathcal{U} $ when \mathcal{F} has duplicates	237

8.3.3	c -Intersection Conflict-Free Set Cover parameterized by k	242
8.4	Matroidal Conflict-free Set Cover	247
8.5	Conclusion	250
9	Fair Vertex Deletion problems	253
9.1	Introduction	253
9.1.1	Previous Work and Deconstructing Hardness	254
9.1.2	Our Results	255
9.2	Preliminaries	258
9.3	Π -FAIR VERTEX DELETION parameterized by treewidth + fairness factor	258
9.4	FAIR q -FORBIDDEN FAMILY VERTEX DELETION parameterized by solution size	265
9.4.1	FPT Algorithm	265
9.4.2	Polynomial Kernel	266
9.4.3	Improved Kernel for FAIR VERTEX COVER parameterized by solution size	271
9.5	Hardness Results	274
9.5.1	W[1]-Hardness for Fair Set	274
9.5.2	NP-Hardness Dichotomy of FAIR VERTEX COVER and FAIR FEEDBACK VERTEX SET	276
9.6	Conclusion	281

IV Conclusion	283
10 Conclusion and Future Directions	285
Bibliography	287

Synopsis

Graphs are one of the most fundamental data structures in computer science because they can be used to model a wide range of problems. As a result, solving optimization problems on graphs is a well-studied area in theoretical computer science. Unfortunately, the majority of such problems are NP-Hard. Hence we do not expect polynomial-time algorithms for such problems. This motivates research on these problems using frameworks where there are trade-offs between the running time of the algorithm and the quality of the solution obtained. One such framework is to restrict the input graph to a well-known graph class. Many of the optimization problems were found to have polynomial time algorithms taking advantage of the structure of the graph arising from such a restriction.

The existence of polynomial time algorithms for problems on a particular graph class Π raises the following question. What if the graph you are looking at does not belong to the graph class Π , but it is not ‘far away’ from one? Specifically, the graph has a few, say k vertices whose deletion results in a graph of graph class Π . If we know that the graph G satisfies such property and also is given the set of vertices $S \subseteq V(G)$ whose deletion results in the graph class Π , we have a good idea of the structure of most of the graph G , specifically $G - S$. Using this structure, fixed parameter tractable (FPT) algorithms parameterized by deletion distance to graph classes Π were obtained for many problems including VERTEX COVER, FEEDBACK VERTEX SET and DOMINATING SET.

This also motivates us to look at the problem of finding such a set of k vertices such that the rest of the graph belongs to some graph class Π . More specifically, the input is of

the form (G, k) and we aim to obtain a set S with $|S| = k$ such that $G - S$ belong to the graph class Π . Such problems are referred to as *vertex deletion problems*. These problems are well-studied with examples including VERTEX COVER, FEEDBACK VERTEX SET, ODD CYCLE TRANSVERSAL, CHORDAL VERTEX DELETION where the graph class Π is edge-less, forest, bipartite and chordal respectively.

In this thesis we investigate vertex deletion problems in the following directions.

Deletion to Scattered Graph Classes. In the first direction, we initiate the study of a natural variation of the problem of deletion to *scattered graph classes*. In this case, we want to delete at most k vertices so that in the resulting graph, each connected component belongs to one of a fixed number of graph classes. Suppose it is the case that each component of a graph belongs to some graph class where a given problem is solvable in polynomial time. Then for most cases, by combining the solutions from each of the components, we can find the solution for the entire graph in polynomial time as well. Finding the vertex deletion set to such collection of components is thus intriguing.

A simple hitting set based approach for this problem is no longer feasible even if each of the graph classes is characterized by finite forbidden sets. Nonetheless, we show the following results.

- We show that this problem is non-uniformly FPT when the deletion problem corresponding to each of the finite classes is known to be FPT and the properties that a graph belongs to any of the classes is expressible in Counting Monodic Second Order (CMSO) logic. The algorithm is based on a result by Lokshtanov et al. [121] that essentially allows us to use the technique of *recursive understanding* in a black-box manner. Unfortunately, the running time of the algorithm has gargantuan constant overheads.
- We provide a $2^{\text{poly}(k)} n^{\mathcal{O}(1)}$ FPT algorithm when each of the graph classes has a finite forbidden set. This algorithm uses the well-known techniques of iterative

compression and important separators, and follows the approach of Ganian et al [76] for a similar problem on constraint satisfaction problems (CSPs).

- Later, we do a deep dive on pairs of specific graph classes (Π_1, Π_2) in which we would like the connected components of the resulting graph to belong to, and design simpler and more efficient FPT algorithms. We design a general algorithm for pairs of graph classes (possibly having infinite forbidden sets) satisfying certain conditions. Our general method covers pairs of graph classes including (Interval, Trees), (Chordal, Bipartite Permutation), (Clique, Planar) and (Clique, Bounded Treewidth) for (Π_1, Π_2) and runs in $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ or $k^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ time depending on the pair of graph classes. Our algorithm makes non-trivial use of the branching technique and as a black box, FPT algorithms for deletion to individual graph classes. We also provide efficient approximation algorithms for deletion to these pairs of graph classes.

Deletion distance parameterizations. In the second direction, we look at the associated optimization problems where the parameter is the deletion distance to a graph class. Generally such problems are studied under the assumption that you are given a deletion set of size k as input. If the problem of finding such a deletion set is known to be FPT, we can use it to find such a set. The running time would be the sum of the running times for finding the set and the algorithm where the modulator is assumed to be given. For most parameterizations, assuming that a modulator is given is reasonable as the corresponding vertex deletion problem has FPT algorithms whose running time is comparable to the time taken by the algorithm designed for the problem given the modulator. But there are cases where the running time for the former is significant enough to pay attention. We examine one such parameter, deletion distance to chordal graphs. For example, the VERTEX COVER problem parameterized by deletion distance to chordal graphs k have a simple $2^k n^{\mathcal{O}(1)}$ algorithm given the modulator. But the current best running time to find such a modulator takes $k^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ time [35].

We develop an algorithmic framework for problems such as VERTEX COVER, FEEDBACK VERTEX SET and ODD CYCLE TRANSVERSAL that either identifies that there is no chordal deletion set of size k or give a $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ algorithm to solve them. We do so by constructing a tree decomposition of the given graph in $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ time where each bag is a union of four cliques and $\mathcal{O}(k)$ vertices and applying standard dynamic programming algorithms over this special tree decomposition. This special tree decomposition can be of independent interest.

We also study deletion distance parameterizations for some other graph problems.

- We look at the $(n - k)$ LIST COLORING problem where we are given a graph with each vertex having a list of size $n - k$ and ask if there is a coloring corresponding to the lists. We show that the problem is FPT parameterized by k . We do so by showing that the problem is FPT if the graph is $f(k)$ vertices away from a clique.
- We give improved FPT algorithms and lower bounds for variants of DOMINATING SET problem in graphs that are k vertices away from a cluster graph or a split graph.

Vertex Deletion with additional constraints. In the third direction, we investigate vertex deletion problems where the deletion set is also required to satisfy additional constraints.

- We consider the case where the deletion set is also required to form an independent set. Such problems are called conflict-free problems and they have already been studied for various problems such as CONFLICT-FREE VERTEX COVER, CONFLICT-FREE FEEDBACK VERTEX SET. We look at the conflict-free version of SET COVER, where there is also a graph on the sets of the input family and we want the set cover to form an independent set in the graph. Specifically, we have a universe \mathcal{U} , a family \mathcal{F} of subsets of \mathcal{U} , a graph $G_{\mathcal{F}}$ with vertex set \mathcal{F} and an integer k and we check if there is a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ of size at most k such that $\cup_{F \in \mathcal{F}'} F = \mathcal{U}$ and \mathcal{F}' forms an independent set in $G_{\mathcal{F}}$. Though SET COVER is not a graph problem, it can be seen as a generalization of various deletion problems such as VERTEX COVER by

associating a family of forbidden graphs incident to each vertex. We look at various parameterizations of CONFLICT-FREE SET COVER under various conditions where the corresponding SET COVER problem is FPT. Here, we also restrict the input graph to belong to popular graph classes and provide upper and lower bounds. We list a few of the results below.

- We give a $f(k)|\mathcal{F}|^{o(k)}$ time lower bound for 1-INTERSECTION CONFLICT-FREE SET COVER assuming the Exponential Time Hypothesis (ETH). The lower bound holds even when $G_{\mathcal{F}}$ is restricted to bipartite graphs where INDEPENDENT SET is polynomial-time solvable.
- We give an FPT algorithm for CONFLICT-FREE SET COVER parameterized by $|\mathcal{U}|$ even in presence of duplicates when we restrict $G_{\mathcal{F}}$ to chordal graphs via dynamic programming on the clique tree decomposition of the graph.
- We also study the CONFLICT-FREE SET COVER problem where there is an underlying (linearly representable) matroid on the family of subsets, and we want the solution to be an independent set in the matroid. We show that the problem FPT when parameterized by the universe size, using the idea of dynamic programming over representative families [71].
- We also study vertex deletion problems where the deletion set is also required to form a fair set. A d -fair set $S \subseteq V(G)$ is such that for every $v \in V(G)$, $|N(v) \cap S| \leq d$.
 - We look at fair vertex deletion problems having a finite forbidden family and provide simple FPT and polynomial kernel results parameterized by solution size.
 - We then focus on FAIR VERTEX COVER which also a finite forbidden family which is $\{K_2\}$. We give a better kernel parameterized by solution size. We also give an FPT algorithm parameterized by treewidth and the fairness factor d for this problem.

- We look at FAIR FEEDBACK VERTEX SET and show that the problem is FPT parameterized by solution size as well as the sum of treewidth and fairness factor.
- Finally, we look at the problem of FAIR SET of whether a d -fair set of size k exist in the graph. We know that the problem is $W[1]$ -hard parameterized by k for $d = 1$ even in 3-degenerate graphs.

List of Figures

1.1	A graph hierarchy of inclusions depicting tractability of some vertex deletion problems	37
3.1	An $X - Y$ tight separator sequence of order two with $U = \emptyset$	79
3.2	The case where X is incomparable with P_1	86
3.3	Example of a marked forbidden set	88
3.4	Example showing paths between vertices of a marked forbidden set . . .	89
3.5	Forest F that provides required connectivities of marked forbidden set vertices. The vertices colored grey correspond to marked vertices and white correspond to other vertices of F . The forest F has a degree two path between x_1 and x_2 with all the internal vertices unmarked. If an unmarked vertex y in this path has an edge to some marked vertex x_3 , then the forest obtained by replacing an edge adjacent to y in the path with (x_3, y) also preserves the connectivities but has an unmarked vertex as a leaf giving a contradiction.	90
3.6	The graph $G' \setminus K^{nr}$ obtained from gluing the graphs $G[R[W_1, P_1]$ and $\hat{G} \setminus K^{nr}$ along P_1^r where K^r is an optimal $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator	92

3.7	A demonstration of how $u_1, u_2 \in C_M$ are connected in the graph $G' \setminus K'$ (denoted by the grey region). We know from the marking procedure that both C and C_M are such that paths between vertices corresponding to u_1 and u_2 have its first and last vertex of P_1^r as t_1 and t_2 . We replace the path between t_2 and u_2' with the path between t_2 and u_2 guaranteed from the forest F	97
4.1	An illustration of a shortest path between a closest (claw, triangle) pair. . .	120
4.2	Obstructions for Graph Classes	128
4.3	An illustration of a shortest path between a closest (long claw, triangle) pair.	129
5.1	Reduction from HITTING SET to VERTEX COVER BY CLSVD	174
6.1	Crown Decomposition	180
6.2	List coloring in $clique + f(k)$	184
7.1	An illustration of the construction described in Theorem 37	209
7.2	Illustration of Branching Rule 7. Note that the number of blue vertices drop by at least two in each of the branches.	213
7.3	Illustration of Branching Rule 8 for the first case. Note that the number of blue vertices in S drops by at least two in each of the branches.	216
7.4	Illustration of Branching Rule 8 for the second case. Note that the number of blue vertices in S drops by two in each of the branches.	216
7.5	In this example, S is a minimum vertex cover of this graph. But there is no minimal vertex cover that contains only x , but not y from S	218

9.1	Construction of FAIR SET instance with $d = 1$ from MULTICOLORED INDEPENDENT SET instance in Theorem 60	275
9.2	Construction of FAIR FEEDBACK VERTEX SET instance with $d = 1$ from 3-SAT instance in Theorem 64	280

Chapter 1

Introduction

Graphs are one of the most fundamental combinatorial structures in computer science. This is because a lot of real-world problems can be modeled using graphs. These problems arise in several areas such as transportation systems, communication networks, bioinformatics, social network analysis and data organization.

As a result, solving optimization problems on graphs is an active research area in theoretical computer science. Unfortunately, most of such problems are found to be NP-hard [77]. Assuming the widely held $P \neq NP$ conjecture, we do not expect polynomial-time algorithms for such problems. However, this does not absolve us from the need of solving such problems as they are essential in real-world applications. This motivates research on these problems using frameworks where there are trade-offs between the running time of the algorithm and the quality of the solution obtained. Notable such frameworks include heuristic algorithms, approximation algorithms, parameterized algorithms, randomized algorithms, genetic algorithms and restricting the input.

We look at the framework where we restrict the input graph for our problem. It is reasonable to assume that most input graphs corresponding to real-world problems has some (possibly hidden) structure manifested in them [92, 64]. For example, suppose we know that the input graph models an assignment problem where all its edges are between a collection

of agents to a set of tasks. We can then infer that such graphs are bipartite. If we identify some structure in all possible input graphs, we only need to solve problems in which the input graph satisfies such a structure.

Motivated by this, we study problems where we restrict the input graph to belong to a well-known graph class. A *graph class* Π is a collection of graphs satisfying a particular property. Many of the optimization problems which are NP-hard on general graphs were found to have polynomial time algorithms when restricting the input to several graph classes. These algorithms take advantage of the structure of the graph arising from such a restriction. As an example, let us focus on the famous VERTEX COVER problem which is NP-hard on general graphs. But the problem can be solved in polynomial time on several graph classes such as trees, bipartite graphs, chordal graphs, claw-free graphs and bounded treewidth graphs [80, 131, 45]. The techniques used to solve VERTEX COVER also vary as we change the underlying graph class. For example, the algorithm for VERTEX COVER on trees uses dynamic programming in bottom-up order of a rooted tree. The algorithm for VERTEX COVER on bipartite graphs uses network flow techniques observing that the size of the maximum matching is equal to the size of the minimum vertex cover in bipartite graphs. The algorithm for VERTEX COVER on chordal graphs is a greedy algorithm using the Perfect Elimination Ordering of a chordal graph.

Deletion Distance Parameterizations. The existence of polynomial time algorithms for problems on a particular graph class Π raises the following question. What if the graph we are interested in does not belong to the graph class Π , but it is not ‘far away’ from one? Specifically, the graph has a few, say k vertices whose deletion results in a graph belonging to graph class Π . In the absence of such vertices, we know that the problem can be solved in polynomial time. Hence we naturally hesitate to throw the towel on finding efficient algorithms for the problem just due to the presence of a few vertices.

The fact that the integer k is ‘small’ is vital here. We do not expect algorithms with running time polynomial in n for problems on graph k vertices away from Π . This is because if

$k = n$, any graph is k vertices away from Π . Hence we are efficiently solving the problem on general graphs where we know that the problem is NP-hard which is unreasonable. Thus, we aim to design an algorithm whose running time ‘grows’ with k . Such algorithms form the basis of parameterized complexity.

In parameterized complexity, an input of size n also comes with a parameter k . The goal is to design an algorithm with running time $f(k) \cdot n^{\mathcal{O}(1)}$ for the problem for a computable function f . Such an algorithm is called a *fixed-parameter tractable* (FPT) algorithm. Similar to NP, there is a complexity class $W[1]$ such that if a problem is known to be $W[1]$ -hard, we do not expect an FPT algorithm for the problem. Another notion in parameterized complexity is *kernelization* where we aim to get an equivalent instance of a given problem whose size is bounded by $g(k)$ for a parameter k and a computable function g . We refer to Section 2.2 of Chapter 2 for formal definitions.

We aim to obtain FPT algorithms for problems parameterized by k when the input graph is k vertices away from a graph class Π . Let us call such parameterizations as *deletion distance parameterizations*. If we know that a graph G satisfies such property and are also given the set of vertices $S \subseteq V(G)$ whose deletion results in Π , we have a good idea of the structure of most of the graph G , specifically $G - S$. Using this structure, FPT algorithms and kernels parameterized by deletion distance to several graph classes were obtained for many problems including VERTEX COVER, FEEDBACK VERTEX SET and DOMINATING SET. The VERTEX COVER problem was studied with the parameter being deletion distance to forests, cluster graphs, split graphs, quasi-forests, pseudo-forests, degree at most 1 graphs, degree at most 2 graphs, bipartite graphs and chordal graphs [72, 124, 93, 64, 122]. The FEEDBACK VERTEX SET problem was studied with the parameter being deletion distance to edge-less graphs, cluster graphs, split graphs, mock-forests, pseudo-forests, bipartite graphs and chordal graphs [123, 97, 122].

Let us look at the example of VERTEX COVER parameterized by the deletion distance to bipartite graphs (size of odd cycle transversal). We assume that we have a graph G

and a subset $S \subseteq V(G)$ with $|S| = k$ such that $G - S$ is bipartite. We give the following $2^k n^{\mathcal{O}(1)}$ algorithm for VERTEX COVER as follows. We guess a subset $S' \subseteq S$ such that for the optimal vertex cover Z of G , we have $Z \cap S = S'$. We now look at the graph $G - S'$. Note that all the edges with one endpoint in $S \setminus S'$ have to be covered by the other endpoint in $V(G) \setminus S$. Hence all such vertices W have to be in the solution Z . We hence arrive at the graph $G - S - W$ after adding W to the solution and removing isolated vertices of S . This graph is bipartite where we can use our known algorithm for VERTEX COVER on bipartite graphs to solve the problem optimally. Overall, the running time is $2^k n^{\mathcal{O}(1)}$ with 2^k overhead coming from going over all possible subsets of S . Note that the only property of bipartite graphs we used in the algorithm is the polynomial time algorithm for VERTEX COVER on such graphs. Hence, we can generalize the above algorithm to any hereditary graph class where VERTEX COVER is solvable in polynomial time.

Deletion distance parameterizations come under the *parameter ecology program* of problems. A parameter of a problem can be many things. The most common parameter studied in parameterized complexity is the size of the solution itself. Earlier work in parameterized complexity mostly focused on the solution size parameter. But later, from the realization that most practical problem inputs have hidden characterizations, the community started to study problems on a plethora of parameters [62, 25, 97, 64]. Such a parameter ecology program aims to see how the complexity of a problem changes for each parameter. The majority of graph parameters studied are deletion distance to graph classes. There are also other graph parameters such as treewidth, dominating set size, chromatic number, maximum clique size, bandwidth and leaves of a minimum spanning tree.

One can also obtain a hierarchy of deletion distance parameters based on the following observation. If a graph class Π_1 is a subclass of another class Π_2 , then the minimum deletion distance to graph class Π_1 is at least as large as the minimum deletion distance to Π_2 . For example, forests form a subclass of chordal graphs. Hence the size of a minimum feedback vertex set is at least the size of a minimum chordal vertex deletion set. Suppose

a problem is FPT parameterized by deletion distance to Π_2 . Then the problem is FPT parameterized by deletion distance to Π_1 since the latter is a larger parameter. Similarly, if a problem is $W[1]$ -hard parameterized by deletion distance to Π_1 , then the problem is also $W[1]$ -hard parameterized by deletion distance to Π_2 since the latter is a smaller parameter.

Vertex Deletion Problems. Deletion distance parameterizations also motivate us to look at another class of problems. In these problems, we aim to find a set of k vertices such that the rest of the graph belongs to some graph class Π . More specifically, the input is of the form (G, k) , and we aim to obtain a set S with $|S| = k$ such that $G - S$ belongs to the graph class Π . Such problems are referred to as *vertex deletion problems*.

Vertex deletion problems come under a more extensive umbrella of well-studied problems in graphs called *graph modification problems*. In such problems, we want to modify a given graph by addition/deletion of vertices or edges to obtain a *simpler* graph.

Let us try to motivate such problems. The primary motivation stems from situations where the input graph has changed slightly for some reason, and we want to find the few outlier vertices or edges whose deletion/addition restores our original graph. This falls under the category of noise detection problems, which are well-studied in computer science. We have already seen how problems are efficiently solvable in several graph classes. Identifying the outliers in the graph helps in designing FPT algorithms parameterized by the size of the outliers.

A graph class is *non-trivial* if the class and its complement contain infinitely many graphs. A graph class is *hereditary* if it is closed under induced subgraphs. A classical work of Lewis and Yannakakis [110] (see also [153]) showed that if vertex deletion results in a graph from a non-trivial hereditary graph class, then the deletion problem is NP-complete. Since the work of Lewis and Yannakakis, the complexity of vertex deletion problems has been studied in various algorithmic paradigms including approximation and parameterized complexity. Vertex deletion problems have been well-studied in parameterized complexity over the last several years. This has led to the discovery of many powerful techniques

such as iterative compression. Examples of such problems include VERTEX COVER [36], CLUSTER VERTEX DELETION [23], SPLIT VERTEX DELETION [49], FEEDBACK VERTEX SET [106], ODD CYCLE TRANSVERSAL [117], PROPER INTERVAL VERTEX DELETION [152], INTERVAL VERTEX DELETION [34], CHORDAL VERTEX DELETION [35], t -TREEWIDTH VERTEX DELETION [70, 102] and PLANAR VERTEX DELETION [96].

It is well-known that any hereditary graph class can be described by a forbidden set of graphs, finite or infinite, that contains all minimal forbidden graphs in the class. It is known that the deletion problem is FPT as long as the resulting hereditary graph class has a finite forbidden set [32]. This is shown by an easy reduction to the BOUNDED HITTING SET problem. A result of Robertson and Seymour [147] showed that if the graph class has a finite forbidden minor characterization, then the deletion problem is FPT. We also know FPT algorithms for specific graph classes defined by infinite forbidden sets such as FEEDBACK VERTEX SET and ODD CYCLE TRANSVERSAL [45]. We still do not have a precise characterization of the class of graphs for which the deletion problem is FPT. Nevertheless, we know that there are graph classes for which the problem is $W[1]$ -hard [113, 84]. The FPT tractability of several vertex deletion problems to graph classes such as permutation graphs, chordal bipartite graphs, AT-free graphs, comparability graphs and co-comparability graphs remain open. See Figure 1.1 for a non-exhaustive hierarchy of inclusions depicting tractability of some vertex deletion problems.

Recently, some stronger versions of deletion problems have also been studied. In these, we delete vertices so that the resulting graph is ‘almost’ in a graph class. More specifically, we aim to delete at most k vertices such that every connected component of the resulting graph is at most ℓ edges away from being a graph in a graph class Π (see [144, 141, 143]). Some examples of Π studied in this stronger version include *forest*, *pseudo-forest* and *bipartite*.

In this thesis, we investigate vertex deletion problems in the following directions.

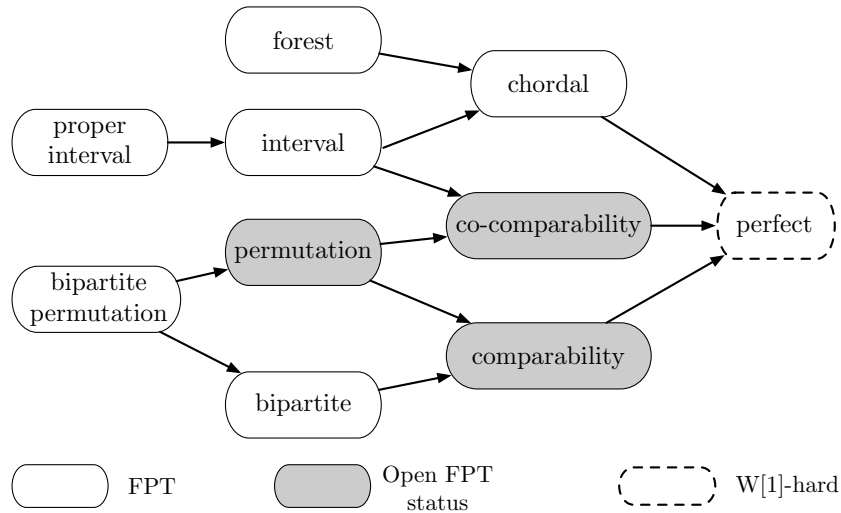


Figure 1.1: A graph hierarchy of inclusions depicting tractability of some vertex deletion problems

1.1 Deletion to Scattered Graph Classes

In the first direction, we initiate the study of a natural variation of the problem of deletion to *scattered graph classes*. In this case, we want to delete at most k vertices so that in the resulting graph, each connected component belongs to one of a fixed number of graph classes. Suppose each component of a graph belongs to some graph class for which the problem of interest is solvable in polynomial time. Then for most problems, by combining the solutions from each of the components, we can find the solution for the entire graph in polynomial time as well.

Let us get back to the example of the VERTEX COVER problem which is polynomial time solvable for 2-treewidth graphs, interval graphs or bipartite graphs. Let Π be the class of graphs whose connected components belong to one of the three graph classes above. Since the three graph classes above are hereditary, the graph class Π is also hereditary. We can independently obtain an optimal vertex cover for each component using the polynomial time VERTEX COVER algorithm for the respective graph class. The union of such solutions is an optimal solution for the entire graph. Thus VERTEX COVER is polynomial time solvable for the graph class Π . This implies that there is a $2^k n^{\mathcal{O}(1)}$ time FPT algorithm for

the problem given the set of k vertices whose deletion results in Π . The problem of finding an optimal vertex deletion set to such scattered graph classes Π is thus intriguing for the same reasons we saw for vertex deletion problems.

Recall that vertex deletion problems can be reduced to an instance of HITTING SET problem leading to a simple branching algorithm on the forbidden graphs of the graph class. Unfortunately, this hitting set idea does not work for the case of deletion to scattered graph classes. For example, let us look at the case where we want the connected components of the resulting graph to be a clique or a biclique (a complete bipartite graph). It is known that cliques forbid exactly P_3 s, the induced paths of length 2, and bicliques forbid P_4 and triangles. So if we want every connected component to be a clique or every connected component to be a biclique, then one can find appropriate constant sized subgraphs in the given graph and branch on them (as one would in a hitting set instance). However, if we want each connected component to be a clique or a biclique, such a simple approach by branching over P_3 , P_4 , or K_3 would not work. Notice that triangles are allowed to be present in clique components and P_3 s are allowed in biclique components. It is not even clear that there will be a finite forbidden set for this resulting graph class.

Nonetheless, we show the following results.

- We show that the deletion problem to scattered graph classes is non-uniformly FPT if it satisfies the following two conditions. First, the deletion problem for each of the finite classes is known to be FPT. Second, the properties that a graph belongs to any of the classes are expressible in Counting Monodic Second Order (CMSO) logic. Our algorithm is based on a result by Lokshtanov et al. [121] which essentially allows us to use the technique of *recursive understanding* in a black-box manner. Unfortunately, the running time of this algorithm has gargantuan constant overheads.
- We provide a $2^{\text{poly}(k)}n^{\mathcal{O}(1)}$ time FPT algorithm when each graph classes has a finite forbidden set. This algorithm uses the well-known techniques of iterative

compression and important separators and follows the approach of Ganian et al [76] for a similar problem on constraint satisfaction problems (CSPs).

- Later, we do a deep dive on deletion to scattered graph classes when the number of classes is exactly two. We do so to obtain simpler and more efficient FPT algorithms. We design a general algorithm for the pairs of graph classes (Π_1, Π_2) (possibly having infinite forbidden sets) satisfying certain conditions. Our general method covers pairs of graph classes including (Interval, Trees), (Chordal, Bipartite Permutation), (Clique, Planar) and (Clique, Bounded Treewidth) for (Π_1, Π_2) . It runs in $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ or $k^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ time depending on the pair of graph classes. Our algorithm makes non-trivial use of the branching technique and as a black box, FPT algorithms for deletion to individual graph classes. We also provide efficient approximation algorithms for deletion to these pairs of graph classes.

1.2 Deletion distance parameterizations

In the second direction, we look at optimization problems where the parameter is the deletion distance to a graph class. Generally, such problems are studied under the assumption that we are given a deletion set of size k as input which we call a *modulator*. If the problem of finding such a deletion set is known to be FPT, we can use it to find such a set. The running time would be the sum of the running times for finding the set and the algorithm where the modulator is assumed to be given.

For most parameterizations, assuming that a modulator is given is reasonable as the corresponding vertex deletion problem has FPT algorithms whose running time is comparable to the time taken by the algorithm designed for the problem given the modulator. For example, let us look at VERTEX COVER parameterized by the size of odd cycle transversal. We have already seen that the problem has a $2^k n^{\mathcal{O}(1)}$ algorithm if an odd cycle transversal set of size k is assumed to be given as input. The current best FPT algorithm for ODD

CYCLE TRANSVERSAL has running time $2.314^k n^{\mathcal{O}(1)}$ [117]. Hence we have an algorithm with $2.314^k n^{\mathcal{O}(1)}$ running time if we do not assume that modulator is given as input. This is not so far from the $2^k n^{\mathcal{O}(1)}$ running time with the assumption that a modulator is given as input.

However, there are cases where the running time for finding the modulator is significant enough to pay attention to. We examine one such parameter, deletion distance to chordal graphs. The VERTEX COVER problem parameterized by deletion distance to chordal graphs k has a simple $2^k n^{\mathcal{O}(1)}$ algorithm given the modulator. But the current best running time to find such a modulator takes $k^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ time [35].

We develop an algorithmic framework for problems such as VERTEX COVER, FEEDBACK VERTEX SET and ODD CYCLE TRANSVERSAL that either identifies that there is no chordal deletion set of size k or give a $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ algorithm to solve them. We do so by constructing a tree decomposition of the given graph in $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ time where each bag is a union of four cliques and $\mathcal{O}(k)$ vertices and applying standard dynamic programming algorithms over this special tree decomposition. This special tree decomposition can be of independent interest.

We also study deletion distance parameterizations for some other graph problems.

- We look at the $(n - k)$ LIST COLORING problem where we are given a graph with each vertex having a list of size $n - k$ and ask if there is a coloring corresponding to the lists. We show that the problem is FPT parameterized by k . We do so by showing that the problem is FPT if the graph is $f(k)$ vertices away from a clique for a function f .
- We give FPT algorithms and lower bounds for several variants of DOMINATING SET problem parameterized by deletion distance to cluster graphs and split graphs.

1.3 Vertex Deletion with additional constraints

In the third direction, we investigate vertex deletion problems where the deletion set is also required to satisfy additional constraints.

Such problems were already studied before. Many problems were studied with the additional condition being the solution set has to form an *independent set* such as INDEPENDENT FEEDBACK VERTEX SET [132], INDEPENDENT ODD CYCLE TRANSVERSAL [128, 118] and INDEPENDENT DOMINATING SET [58]. Another set of problems is where the graph induced by the solution set is *connected* such as CONNECTED VERTEX COVER [47], CONNECTED FEEDBACK VERTEX SET [133], CONNECTED ODD CYCLE TRANSVERSAL [51] and CONNECTED DOMINATING SET [66]. The problems with independent set constraints can be generalized to a set of problems called *conflict-free* problems where there is another graph H with the same vertex set and the solution has to form an independent set in H [90, 1]. This can be further generalized to *simultaneous* problems where the solution set in G has to satisfy a property π in H [4].

A common theme for many vertex deletion problems when adding constraints is that the techniques such as reduction and branching rules that worked for the normal deletion problem are no longer applicable. We must cleverly devise rules modifying the existing rules so that the constraint requirement is also satisfied by the reduced instance. We will see similar rules related to the results in this direction.

- We look at conflict-free problems where the deletion set is also required to form an independent set in an input graph. We look at the conflict-free version of SET COVER where there is also a graph on the sets of the input family, and we want the set cover to form an independent set in the graph. Specifically, we have a universe \mathcal{U} , a family \mathcal{F} of subsets of \mathcal{U} , a graph $G_{\mathcal{F}}$ with vertex set \mathcal{F} and an integer k and we check if there is a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ of size at most k such that $\cup_{F \in \mathcal{F}'} F = \mathcal{U}$ and \mathcal{F}'

forms an independent set in $G_{\mathcal{F}}$. Though SET COVER is not a problem on graphs, it can be seen as a generalization of various deletion problems such as VERTEX COVER by associating a family of forbidden graphs incident to each vertex. We look at various parameterizations of CONFLICT-FREE SET COVER under various conditions where the corresponding SET COVER problem is FPT. Here, we also restrict the input graph to popular graph classes and provide upper and lower bounds. We list a few of the results below.

- We give a $f(k)|\mathcal{F}|^{o(k)}$ time lower bound for 1-INTERSECTION CONFLICT-FREE SET COVER assuming the Exponential Time Hypothesis (ETH). The lower bound holds even when $G_{\mathcal{F}}$ is restricted to bipartite graphs where INDEPENDENT SET is polynomial-time solvable.
- We give an FPT algorithm for CONFLICT-FREE SET COVER parameterized by $|\mathcal{U}|$ even in the presence of duplicates when we restrict $G_{\mathcal{F}}$ to chordal graphs via dynamic programming on the clique tree decomposition of the graph.
- We also study the CONFLICT-FREE SET COVER problem where there is an underlying (linearly representable) matroid on the family of subsets, and we want the solution to be an independent set in the matroid. We show that the problem is FPT when parameterized by the universe size, using the idea of dynamic programming over representative families [71].
- We also study vertex deletion problems where the deletion set is also required to form a fair set. A d -fair set $S \subseteq V(G)$ is such that for every $v \in V(G)$, $|N(v) \cap S| \leq d$.
 - We look at fair vertex deletion problems having a finite forbidden family and provide simple FPT and polynomial kernel results parameterized by solution size.
 - We then focus on FAIR VERTEX COVER which also has a finite forbidden family which is $\{K_2\}$. We give a better kernel parameterized by solution size.

We also give an FPT algorithm parameterized by treewidth and the fairness factor d for this problem.

- We then look at FAIR FEEDBACK VERTEX SET and show that the problem is FPT parameterized by solution size as well as the sum of treewidth and fairness factor.
- We look at the problem of FAIR SET of whether a d -fair set of size k exists in the graph. We know that the problem is $W[1]$ -hard parameterized by k for $d = 1$ even in 3-degenerate graphs.
- Finally, we complete the NP-hardness dichotomy of FAIR VERTEX COVER and FAIR FEEDBACK VERTEX SET problems for all values of fairness factor d . We show that FAIR VERTEX COVER is polynomial-time solvable when $d = 1$ or $d = 2$ and NP-hard otherwise. We then show that FAIR FEEDBACK VERTEX SET is NP-hard for every integer $d \geq 1$.

1.4 Organization of Thesis

This thesis is organized into five parts.

1. The first part contains the introduction of the thesis and preliminaries on parameterized complexity, graph theory, second-order logic and matroids. This contains the first two chapters.
2. In the second part, we look at vertex deletion problems for scattered graph classes. In Chapter 3, we design FPT algorithms for deletion problems for scattered graph classes satisfying some general conditions. We also design a faster algorithm for the case when the forbidden families corresponding to each of the graph classes are finite. In Chapter 4, we look at the case where there are only pairs of scattered graph classes and design much faster FPT algorithms and also approximation algorithms

for problems satisfying certain conditions.

3. In the third part, we give FPT and kernel results for problems where the parameter is deletion distance to some graph class. In Chapter 5, we look at deletion distance parameterizations where we assume that the modulator is not given as input. In Chapter 6, we give FPT algorithms for $n - k$ LIST COLORING. Finally, in Chapter 7, we look at several variants of the DOMINATING SET problem parameterized by deletion distance to cluster and split graphs.
4. In the fourth part, we give FPT and kernel results for vertex deletion problems required to satisfy some additional constraints. This includes CONFLICT-FREE SET COVER which is covered in Chapter 8 and FAIR VERTEX DELETION problems covered in Chapter 9.
5. In the fifth part, we state our conclusions from the thesis and associated open problems.

Chapter 2

Preliminaries

2.1 Sets, Numbers and some Notations

We use \mathbb{N} to denote the set of natural numbers starting from 0. Given $r \in \mathbb{N}$, we use $[r]$ to denote the set $\{1, \dots, r\}$. Given a finite set A , we use 2^A to denote the family of all subsets of A . For an integer t , we use $\binom{A}{t}$ and $\binom{A}{\leq t}$ to denote the family of all subsets of A size t and at most t respectively.

We use $A \uplus B$ to denote the set formed from the union of disjoint sets A and B . For a function $w : X \rightarrow \mathbb{R}$, we use $w(D) = \sum_{x \in D} w(x)$. We use $\log k$ to denote $\log_2 k$.

For a function $f : \mathbb{N} \rightarrow \mathbb{N}$, we use $\mathcal{O}^*(f(k))$ to denote $\mathcal{O}(f(k)p(n))$ where $p(n)$ is some polynomial in n . This allows us to use \mathcal{O}^* notation to suppress the polynomial factors in the running time of algorithms.

Throughout the thesis, ω denotes the matrix multiplication exponent.

2.2 Parameterized Complexity and Kernelization

A parameterized problem L is a subset of $\Sigma^* \times \mathbb{N}$ for some finite alphabet Σ . An instance of a parameterized problem is denoted by (x, k) where $x \in \Sigma^*, k \in \mathbb{N}$. We assume that k is given in unary and without loss of generality $k \leq |x|$.

Definition 2.2.1 (Fixed-Parameter Tractability). *A parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ is said to be fixed-parameter tractable (FPT) if there exists an algorithm \mathcal{A} , a computable function $f : \mathbb{N} \rightarrow \mathbb{N}$ and a constant c independent of $f, k, |x|$, such that given input (x, k) , runs in time $f(k)|x|^c$ and correctly decides whether $(x, k) \in L$ or not.*

Definition 2.2.2 (Non-Uniformly FPT). *A parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ is said to be non-uniformly FPT if there exists a fixed integer d such that for every fixed $k \in \mathbb{N}$, there exists an algorithm \mathcal{A} that determines whether $(x, k) \in L$ in $O(|x|^d)$ time (Hence, there is a different algorithm for each value of k).*

A closely related notion to fixed-parameter tractability is the notion of kernelization.

Definition 2.2.3 (Parameterized Reduction). *Let $P_1, P_2 \in \Sigma^* \times \mathbb{N}$ be two parameterized languages. Suppose there exists an algorithm \mathcal{B} that takes input (I, k) (an instance of P_1) and constructs an instance (I', k') of P_2 such that the following conditions are satisfied.*

- (I, k) is YES-INSTANCE if and only if (I', k') is YES-INSTANCE.
- $k' \in f(k)$ for some function depending only on k .
- Algorithm \mathcal{B} runs in $g(k)|I|^{\mathcal{O}(1)}$ time.

Then we say that there exists a parameterized reduction from P_1 to P_2 .

Definition 2.2.4 (Kernelization). *Let $L \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized language. Kernelization is a procedure that replaces the input instance (I, k) by a reduced instance (I', k') such that*

- $k' \leq f(k)$, $|I'| \leq g(k)$ for some function f, g depending only on k .
- $(I, k) \in L$ if and only if $(I', k') \in L$.
- The reduction from (I, k) to (I', k') must be computable in $\text{poly}(|I| + k)$ time.

If $g(k) = k^{\mathcal{O}(1)}$ then we say that L admits a polynomial kernel.

We now define a notion of reduction rule that is useful in designing kernels.

Definition 2.2.5 (Reduction Rule). *Let $L \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized language. A reduction rule is a procedure computable in $\text{poly}(|I| + k)$ time that replaces the input instance (I, k) by a reduced instance (I', k') such that $(I, k) \in L$ if and only if $(I', k') \in L$.*

The property of the reduction rule that it translates an instance to an equivalent one, i.e. $(I, k) \in L$ if and only if $(I', k') \in L$ is called the safeness of the reduction rule.

W-hierarchy: In order to capture parameterized languages being FPT or not, the W-hierarchy is defined as $\text{FPT} \subseteq \text{W}[1] \subseteq \dots \subseteq \text{XP}$. It is believed that this subset relation is strict. Hence a parameterized language that is hard for some complexity class above FPT is unlikely to be FPT. If a parameterized language $L \subseteq \Sigma^* \times \mathbb{N}$ can be solved by an algorithm running in $\mathcal{O}(n^{f(k)})$ time, then we say $L \in \text{XP}$. In such a situation we also say that L admits an XP algorithm.

Definition 2.2.6 (para-NP-hardness). *A parameterized language $L \subseteq \Sigma^* \times \mathbb{N}$ is called para-NP-hard if it is NP-hard for some constant value of the parameter.*

It is believed that a para-NP-hard problem is not expected to admit an XP algorithm as otherwise it will imply $\text{P} = \text{NP}$.

Theorem 1. *Let there exist a parameterized reduction from parameterized problem P_1 to parameterized problem P_2 . Then if P_2 is fixed-parameter tractable then so is P_1 . Equivalently if P_1 is $\text{W}[i]$ -hard for some $i \geq 1$, then so is P_2 .*

It is well-known [45] that a decidable parameterized problem is fixed-parameter tractable if and only if it has a kernel. But the kernel size could be exponential (or worse) in the parameter. There is a hardness theory for problems having a polynomial sized kernel. Towards that, we define the notion of polynomial parameter transformation.

Definition 2.2.7 (Polynomial parameter transformation (PPT)). *Let P_1 and P_2 be two parameterized languages. We say that P_1 is polynomial parameter reducible to P_2 if there exists a polynomial time computable function (or algorithm) $f : \Sigma^* \times \mathbb{N} \rightarrow \Sigma^* \times \mathbb{N}$, a polynomial $p : \mathbb{N} \rightarrow \mathbb{N}$ such that $(x, k) \in P_1$ if and only if $f(x, k) \in P_2$ and $k' \leq p(k)$ where $f((x, k)) = (x', k')$. We call f to be a polynomial parameter transformation from P_1 to P_2 .*

The following proposition gives the use of the polynomial parameter transformation for obtaining kernels for one problem from another.

Proposition 2.2.1 ([17]). *Let $P, Q \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems and assume that there exists a PPT from P to Q . Furthermore, assume that the classical version of P is NP-hard and Q is in NP. Then if Q has a polynomial kernel, P has a polynomial kernel.*

We use the following conjectures and theorems to prove some of our lower bounds.

Conjecture 2.2.1 (Strong Exponential Time Hypothesis (SETH)). ([88]) *There is no $\varepsilon > 0$ such that $\forall q \geq 3$, q -CNFSAT can be solved in $\mathcal{O}^*((2 - \varepsilon)^n)$ time where n is the number of variables in input formula.*

Conjecture 2.2.2 (Exponential Time Hypothesis (ETH)). ([88, 87]) *3-CNF-SAT cannot be solved in $\mathcal{O}^*(2^{o(n)})$ time where the input formula has n variables and m clauses.*

Conjecture 2.2.3 (Set Cover Conjecture (SCC)). ([48]) *There is no $\varepsilon > 0$ such that SET COVER can be solved in $\mathcal{O}^*((2 - \varepsilon)^n)$ time where n is the size of the universe.*

Theorem 2 ([56]). *SET COVER parameterized by the universe size does not admit any polynomial kernel unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.*

Theorem 3 ([75]). *CNF-SAT parameterized by the number of variables admits no polynomial kernel unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.*

For more details of parameterized complexity, we refer to [45].

2.3 Graph Theory

2.3.1 Basic Notations and Definitions

We use standard graph theoretic terminology from Diestel's book [55]. For a graph $G = (V, E)$, we denote $n = |V(G)|$ and $m = |E(G)|$. For a connected component C of G , we often abuse the notation to denote C as also the vertex set of C . For a set $X \subseteq G$, we use $G[X]$ to denote the graph induced on the vertex set X and we use $G - X$ (or $G \setminus X$) to denote the graph induced by the vertex set $V(G) \setminus X$.

For $V' \subseteq V$, $G[V']$ and $G - V'$ denote the graph induced on V' and $V \setminus V'$, respectively. We also use $G \setminus V'$ to denote the graph induced on vertex set $V \setminus V'$. For a vertex $v \in V$, $G - v$ denotes the graph $G - \{v\}$.

The complement \bar{G} of a graph G is the graph on $V(G)$ where two distinct vertices of \bar{G} are adjacent if and only if they are not adjacent in G . Two vertices u and v of a graph G are neighbors (non-neighbors) if and only if $(u, v) \in E(G)$ ($(u, v) \notin E(G)$). For any vertex $v \in V(G)$, we denote the set of neighbors (non-neighbors) of v by $N_G(v)$ ($\bar{N}_G(v)$) or briefly by $N(v)$ ($\bar{N}(v)$). For any $W \subseteq V(G)$ we define $N_G(W) = \{\cup_{v \in W} N_G(v)\}$ ($\bar{N}_G(W) = \{\cup_{v \in W} \bar{N}_G(v)\}$).

For a vertex $v \in V(G)$, we denote $N_G(v) = \{(u \in V(G)) | (u, v) \in E(G)\}$ as the open neighborhood of v . When there is no confusion, we drop the subscript G . By $N[v]$ we denote the close neighborhood of v , i.e. $N[v] = N(v) \cup \{v\}$. For a $S \subseteq V(G)$, we denote $N(S) = \{v \in V(G) | \exists u \in S \text{ such that } (u, v) \in E(G)\} \setminus S$. And we denote $N[S] = N(S) \cup S$.

By $N^{=2}(v)$, we denote the set of vertices that are at minimum distance exactly two from v . We say that for vertices $u, v \in V$, u dominates v if $v \in N(u)$.

A set $S \subseteq V(G)$ is called an *independent set* if for all $u, v \in S$, $(u, v) \notin E(G)$. Similarly for a subset $S \subseteq V(G)$, $G[S]$ is called a *clique* if for every $u, v \in S$, $(u, v) \in E(G)$. For $\ell \in \mathbb{N}$, we use P_ℓ to denote the path on ℓ vertices. A vertex t is *reachable* from a vertex s if there exists a path in the graph which starts with s and ends with t . A *tree* is a connected graph with no cycles. A *forest* is a graph, every connected component of which is a tree.

A graph G is called a *bipartite graph* if there exists a partition of $V(G) = A \uplus B$ such that for every edge $uv \in E(G)$, $u \in A$ and $v \in B$. A *cluster graph* is a graph where every connected component forms a clique. A graph G is called a *split graph* if its vertex set can be partitioned into two parts $V(G) = C \uplus I$ such that C is a clique and I is an independent set. A graph is called a *cactus graph* if every edge of the graph is contained in at most one cycle. Let A be a set of three arbitrary vertices of a graph G . Then, A is called an *asteroidal triple (AT)* if between every two vertices of A , there is a path avoiding the third vertex. A graph is called a *chordal graph* if it has no induced cycle of length at least four. A graph is called an *interval graph* if it is chordal and AT-free. Alternatively, any interval graph has an interval representation. It means that every vertex of an interval graph can be represented as an interval on the real line and two vertices are adjacent if and only if the intervals representing the corresponding vertices intersect. A graph is called a *proper interval graph* if it is an interval graph with an interval representation such that no interval properly contains any other interval. A graph is called a *bipartite permutation graph* if it is bipartite and AT-free. The *arboricity* of a graph is the minimum number of forests into which its edges can be partitioned.

A *k-subdivision* of a graph G is the graph created from G by subdividing every edge by exactly k vertices. A graph class is a *somewhere dense* graph class when there exists a threshold t such that every complete graph appears as a t -subdivision in a subgraph of a graph in the class. On the contrary, if such a threshold does not exist, the class is *nowhere*

dense. We refer to [135] for more details on nowhere dense graphs.

We say that a subset $Z \subseteq V(G)$ *disconnects* a subset $S \subseteq V(G)$ if there exists $v, w \in S$ with $v \neq w$ such that v and w occur in different connected components of the graph $G \setminus Z$. We call a path P in a graph G as a *degree 2 path* if all the internal vertices of the path have degree 2 in G .

Let $u, v \in V(G)$. We use $d_G(u, v)$ to denote the length of a *shortest path* from u to v in G . For $P, Q \subseteq V(G)$, we define $d_G(P, Q) = \min_{u \in P, v \in Q} \{d_G(u, v)\}$.

We say that a graph G is a union of ℓ cliques if $V(G) = V_1 \uplus \dots \uplus V_\ell$ and V_i is a clique in G for all $i \in \{1, \dots, \ell\}$.

A graph G is k -colorable if its vertices can be colored in such a way that the endpoints of every edge of G have two different colors.

A graph $G = (V, E)$ is said to be d' -*degenerate* if there exists an ordering of V such that each vertex has at most d' neighbors to its right in the ordering. The minimum such possible d' a graph can have is called its *degeneracy*.

2.3.2 Graph Separators

Definition 2.3.1 (Separator and Separation). *Given a graph G and vertex subsets $A, B \subseteq V(G)$, a subset $C \subseteq V(G)$ is called a separator of A and B if every path from a vertex in A to a vertex in B (we call it $A - B$ path) contains a vertex from C . A pair of vertex subsets (A, B) is a separation in G if $A \cup B = V(G)$ and $A \cap B$ is a separator of $A \setminus B$ and $B \setminus A$.*

Definition 2.3.2 (Balanced Separator and Balanced Separation). *For a graph G , a weight function $w : V(G) \rightarrow \mathbb{R}_{\geq 0}$ and $0 < \alpha < 1$, a set $S \subseteq V(G)$ is called an α -balanced separator of G with respect to w if for any connected component C of $G - S$, $w(V(C)) \leq \alpha \cdot w(V(G))$. A pair of vertex subsets (A, B) is an α -balanced separation in G with respect to w if (A, B) is a separation in G and $w(A \setminus B) \leq \alpha \cdot w(V(G))$ and $w(B \setminus A) \leq \alpha \cdot w(V(G))$. Note that*

the sets A and B could be empty. It could also be that $A \subseteq B$ or vice-versa as well.

We borrow the following definitions from [126], [76].

Definition 2.3.3. Let G be a graph and disjoint subsets $X, S \subseteq V(G)$. We denote by $R_G(X, S)$ the set of vertices that is reachable from $X \setminus S$ in the graph $G \setminus S$. We denote $R_G[X, S] = R_G(X, S) \cup S$. Finally we denote $NR_G(X, S) = V(G) \setminus R_G[X, S]$ and $NR_G[X, S] = NR_G(X, S) \cup S$. We drop the subscript G if it is clear from the context.

Definition 2.3.4. [126] Let G be a graph and $X, Y \subseteq V(G)$.

- A vertex set S disjoint from X and Y is said to disconnect X and Y if $R_G(X, S) \cap Y = \emptyset$. We say that S is an $X - Y$ **separator** in the graph G .
- An $X - Y$ separator is **minimal** if none of its proper subsets is an $X - Y$ separator.
- An $X - Y$ separator S_1 is said to **cover** an $X - Y$ separator S with respect to X if $R(X, S) \subset R(X, S_1)$.
- Two $X - Y$ separators S_1 and S_2 are said to be **incomparable** if neither covers the other.
- In a set \mathcal{H} of $X - Y$ separators, a separator S is said to be **component-maximal** if there is no separator S' in \mathcal{H} which covers S . Component-minimality is defined analogously.
- An $X - Y$ separator S_1 is said to **dominate** an $X - Y$ separator S with respect to X if $|S_1| \leq |S|$ and S_1 covers S with respect to X .
- We say that S is an **important** $X - Y$ separator if it is minimal and there is no $X - Y$ separator dominating S with respect to X .

2.3.3 Tree Decomposition and Treewidth

Definition 2.3.5 (Tree decomposition). Given a graph $G = (V, E)$, a tree decomposition is a pair (X, T) , where $X = \{X_1, \dots, X_n\}$ is a family of subsets of V , and T is a tree whose nodes are the subsets X_i , satisfying the following properties:

- $\bigcup_{i=1}^n X_i = V$.
- For all edges $(u, v) \in E$, there is a subset X_i that contains both u and v .
- If X_i and X_j both contain a vertex v , then all nodes X_k of the tree in the (unique) path between X_i and X_j contain v as well.

The sets X_i are called the bags corresponding to node i .

Definition 2.3.6 (Treewidth). The width of tree decomposition (X, T) equals $\max_{t \in V(T)} |X_t| - 1$. The treewidth of a graph G is the minimum possible width of a tree decomposition of G .

Definition 2.3.7 (Nice Tree decomposition). A nice tree decomposition is a tree decomposition T satisfying the following properties:

- Let us arbitrarily root the tree T . For the root of the tree r , $X_r = \emptyset$.
- $X_l = \emptyset$ for all the leaf nodes of the tree.
- Every other node of T are one of three types:
 - Introduce Node: A node i with exactly one child j such that $X_i = X_j \cup \{v\}$ for some vertex $v \notin X_j$.
 - Forget Node: A node i with exactly one child j such that $X_i = X_j \setminus \{v\}$ for some vertex $v \in X_j$.
 - Join Node: A node i with exactly two children j and j' such that $X_i = X_j = X_{j'}$.

Lemma 2.3.1 (Lemma 7.4 of [45], Lemma 13.1.3 of [103]). *Given a tree decomposition (X, T) of a graph G of width at most k , in polynomial time one compute a nice tree decomposition (X', T') of G of width at most k that has at most $\mathcal{O}(k|V(G)|)$ nodes. Moreover, for each $t' \in V(T')$, there is a $t \in V(T)$ such that $X_{t'} \subseteq X_t$.*

2.4 Second Order Logic

Definition 2.4.1 (Monadic Second Order logic). *Monadic second-order logic (MSO) is a logic with two types of quantifiers: one can quantify over elements, and one can quantify over sets of elements.*

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

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

Counting Monadic Second Order Logic (CMSO) extends MSO by including atomic sentences testing whether the cardinality of a set is equal to q modulo r , where q and r are integers such that $0 \leq q < r$ and $r \geq 2$. That is, CMSO is MSO with the following

atomic sentence: $\text{card}_{q,r}(S) = \text{true}$ if and only if $|S| \equiv q \pmod r$, where S is a set. We refer to [44, 9] for a detailed introduction to CMSO.

2.5 Matroids

Definition 2.5.1 (Matroid). A matroid M is a pair (E, \mathcal{I}) where E is the ground set and \mathcal{I} is the family of subsets of E (called the independent sets of M) satisfying the following properties:

- $\emptyset \in \mathcal{I}$.
- If $A' \subseteq A$ and $A \in \mathcal{I}$, then $A' \in \mathcal{I}$.
- If $A, B \in \mathcal{I}$ and $|A| < |B|$, then there exists an $e \in B \setminus A$ such that $A \cup \{e\} \in \mathcal{I}$.

Definition 2.5.2 (Rank of a matroid). For a matroid $M = (E, \mathcal{I})$, an inclusion-wise maximal set of \mathcal{I} is called a basis of the matroid. It can be shown that all the bases of a matroid have the same size. This size is called the rank of the matroid M .

Definition 2.5.3 (Linear Matroid). Let A be a matrix over a field \mathbb{F} and let E be the set of columns of A . We define a matroid $M = (E, \mathcal{I})$ as follows: A subset $X \subseteq E$ is an independent set of M if and only if the corresponding columns of A are linearly independent over \mathbb{F} . The matroids that can be defined by such a construction over some field \mathbb{F} are called linear matroids. The matrix A corresponding to the matroid M is called the linear representation of M .

For more details on matroids, we refer to [137].

Part I

Deletion to Scattered Graph Classes

Chapter 3

FPT algorithms for general cases

In this chapter, we address the complexity of a very natural variation of the graph deletion problem, where in the resulting graph, each connected component belongs to one of the finitely many graph classes. In this case, we say that the graph belongs to a *scattered graph class*. Let us formalize the definition below.

Definition 3.0.1 ((Π_1, \dots, Π_d) scattered graph class). *The collection of graphs such that each of its components belongs to at least one of the graph classes Π_i for $i \in [d]$ is called a (Π_1, \dots, Π_d) scattered graph class.*

For example, if each connected component of a graph is at least one of interval, bipartite or claw-free graph, we say that the graph belongs to (Interval, Bipartite, Claw-free) scattered graph class.

Let us now formally define the deletion problem below where we want every connected component of the resulting graph to belong to at least one of the graph classes Π_i with $i \in [d]$ for some finite integer d .

$(\Pi_1, \Pi_2, \dots, \Pi_d)$ VERTEX DELETION

Input: An undirected graph $G = (V, E)$, an integer k , and d graph classes Π_1, \dots, Π_d .

Parameter: k

Question: Is there a subset $Z \subseteq V(G)$, $|Z| \leq k$ such that every connected component of $G - Z$ is in at least one of the graph classes Π_1, \dots, Π_d ?

We call a set Z a $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -*modulator* if every connected component of $G \setminus Z$ is in one of the graph classes Π_i for $i \in d$.

We look at the case when each problem Π_i VERTEX DELETION is known to be FPT and the property that “graph G belongs to Π_i ” is expressible in CMSO logic (See Section 2.4 for formal definitions). We call this problem INDIVIDUALLY TRACTABLE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ VERTEX DELETION. We show that this problem is non-uniformly¹ fixed parameter tractable.

Theorem 4. INDIVIDUALLY TRACTABLE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ VERTEX DELETION is non-uniformly FPT with respect to solution size k .

The problem INDIVIDUALLY TRACTABLE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ VERTEX DELETION covers a wide variety of collections of popular graph classes. Unfortunately, the running time of the algorithm from Theorem 4 has gargantuan constant overheads. Hence we look at the special case of INDIVIDUALLY TRACTABLE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ VERTEX DELETION named FINITE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ VERTEX DELETION where each of the graph classes is characterized by a finite forbidden set. We get a faster FPT algorithm for FINITE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ VERTEX DELETION using the well-known techniques in parameterized complexity – iterative compression and important separators.

Theorem 5. FINITE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ VERTEX DELETION can be solved in time $2^{\text{poly}(k)} n^{\mathcal{O}(1)}$.

Here, $\text{poly}(k)$ denotes a polynomial in k .

¹See Definition 2.2.2 in Section 2.2

Previous Work: While there has been a lot of work on graph deletion and modification problems, one work that comes close to ours is the work by Ganian, Ramanujan and Szeider [76] where they consider the parameterized complexity of finding strong backdoors to a scattered class of CSP instances. In fact, in their conclusion, they remark that

‘graph modification problems and in particular the study of efficiently computable modulators to various graph classes has been an integral part of parameterized complexity and has led to the development of several powerful tools and techniques. We believe that the study of modulators to ‘scattered graph classes’ could prove equally fruitful and, as our techniques are mostly graph based, our results as well as techniques could provide a useful starting point towards future research in this direction’.

Our work is a starting point in addressing the parameterized complexity of the problem they suggest.

Our Techniques: The FPT algorithm for INDIVIDUALLY TRACTABLE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ VERTEX DELETION in Theorem 4 is based on a result by Lokshtanov et al. [121] that essentially allows us to use the technique of *recursive understanding* in a black-box manner. The result allows one to obtain a non-uniformly FPT algorithm for CMSO-expressible graph problems by designing an FPT algorithm for the problem on a well-connected class of graphs called unbreakable graphs. For the latter, using the observation that only one connected component after deleting the solution is large, which belongs to some particular class Π_i , we use the FPT algorithm for Π_i -VERTEX DELETION to obtain a modulator to the graph class Π_i of size $s(k)$ for a function s . Then we use a branching rule to remove the components in the modulator that are not in Π_i , thereby “revealing” the solution to the problem. For more information, see Section 3.1.

The FPT algorithm of Theorem 5 for the problem FINITE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ VERTEX DELETION is now briefly summarized. We can assume that we have a solution W of size $k + 1$ for our problem using the standard iterative compression technique. Depending on whether or not the solution disconnects W , the problem can be divided into two cases. If it

does not, a simple algorithm based on branching on vertices of the finite forbidden graphs plus some important separators [126] of the graph solves the problem. Otherwise, we can assume that the solution contains a “special” important separator. In this case, we develop a recursive procedure to find a set \mathcal{R} of $2^{\text{poly}(k)}$ vertices, at least one of which intersects the solution. As a result, we devise a branching rule on the vertices of \mathcal{R} to solve the problem. See Section 3.2 for more details.

In the recursive procedure to generate \mathcal{R} , the graph is created by gluing a graph of $\text{poly}(k)$ vertices to an induced subgraph of G along with a set of ‘boundary’ vertices. The techniques we use here are very similar to those used by Ganian et al. [76], in which they find backdoors to a collection of easy Constraint Satisfaction Problems (CSPs). FPT algorithms are developed using similar techniques involving tight separator sequences for problems such as PARITY MULTIWAY CUT [119], DIRECTED FEEDBACK VERTEX SET [120], SUBSET ODD CYCLE TRANSVERSAL [116] and SAVING CRITICAL NODES WITH FIREFIGHTERS [39].

There is a crucial distinction between our algorithm and the problems listed above that were solved using similar techniques. In the latter, the addition and removal of edges and vertices in some ways does not disrupt the problem input. This is used to create gadgets that preserve certain properties in the recursive input graph such as connectivity and parity of paths between pairs of vertices. In FINITE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ VERTEX DELETION, we do not assume any structure for the forbidden graphs corresponding to the graph classes Π_i for $i \in [d]$. As a result, we are not permitted to add or remove edges or vertices as doing so may create or destroy forbidden graphs corresponding to graph classes, thereby changing the problem instance. In order to avoid this issue, we only perform contractions of large degree two paths in the graph up to a certain constant.

3.1 Deletion to scattered classes when each class is individually tractable

We first formally define the problem.

INDIVIDUALLY TRACTABLE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ VERTEX DELETION

Input: An undirected graph $G = (V, E)$, an integer k , and d hereditary graph classes Π_1, \dots, Π_d such that for all $i \in [d]$, Π_i VERTEX DELETION is FPT and properties $P_i(H)$ for input graph H is CMSO expressible.

Parameter: k

Question: Is there a subset $Z \subseteq V(G)$, $|Z| \leq k$ such that every connected component of $G - Z$ is in at least one of the graph classes Π_1, \dots, Π_d ?

We recall the notion of unbreakable graphs from [121].

Definition 3.1.1. A graph G is (s, c) -unbreakable if there does not exist a partition of the vertex set into three sets X, C and Y such that (a) C is an (X, Y) -separator: there are no edges from X to Y in $G \setminus C$, (b) C is small: $|C| \leq c$, and (c) X and Y are large: $|X|, |Y| \geq s$.

We now use the following theorem from [121] which says that if the problem is FPT in unbreakable graphs, then the problem is FPT in general graphs. Let $\text{CMSO}[\psi]$ denote the problem with graph G as an input, and the objective is to determine whether G satisfies ψ .

Theorem 6. [121] Let ψ be a CMSO sentence. For all $c \in \mathbb{N}$, there exists $s \in \mathbb{N}$ such that if there exists an algorithm that solves $\text{CMSO}[\psi]$ on (s, c) -unbreakable graphs in time $\mathcal{O}(n^d)$ for some $d > 4$, then there exists an algorithm that solves $\text{CMSO}[\psi]$ on general graphs in time $\mathcal{O}(n^d)$.

In the following lemma, we show that our problem is CMSO expressible.

Lemma 3.1.1. INDIVIDUALLY TRACTABLE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ VERTEX DELETION is CMSO expressible.

Proof. We use $\text{conn}(X)$ which verifies that a subset X of a graph G induces a connected subgraph. It is known that $\text{conn}(X)$ is expressible by an MSO formula [45]. Also for $X \subseteq V(G)$, we can express the sentence “ $|X| = k$ ” as $\exists x_1, \dots, x_k \forall u \in V(G) (u \in X \implies (\bigvee_{i \in [k]} u = x_i))$

Recall that $P_i(G)$ denotes the graph property “graph G is in Π_i ” for $i \in [d]$ and input graph G . Let the CMSO sentences for properties $P_i(G)$ be $\psi_i(G)$. The overall CMSO sentence for our problem ψ is $\exists X \subseteq V(G), |X| = k, \forall C \subseteq V(G) \setminus X : \text{conn}(C) \implies (\bigvee_{i \in [d]} \psi_i(G[C]))$. \square

Hence Theorem 6 and Lemma 3.1.1 allow us to focus on solving our problem for unbreakable graphs.

Theorem 7. INDIVIDUALLY TRACTABLE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ VERTEX DELETION is FPT in $(s(k), k)$ -unbreakable graphs for any function s of k .

Proof. Let G be an $(s(k), k)$ -unbreakable graph and X be a solution of size k . Look at the connected components of $G - X$. Since X is a separator of size at most k , at most one connected component of $G - X$ has size more than $s(k)$.

Let us first look at the case where no connected component of $G - X$ has size more than $s(k)$. In this case, we can bound the number of connected components by $2s(k)$. Suppose not. Then we can divide vertex sets of connected components into two parts C_1 and C_2 , each having at least $s(k)$ vertices. Then the partition (C_1, X, C_2) of $V(G)$ contradicts that G is $(s(k), k)$ -unbreakable.

Since each component has size at most $s(k)$, we have $|V(G) \setminus X| \leq 2(s(k))^2$. Hence $|V(G)| \leq 2(s(k))^2 + k$. We can solve the problem by going over all subsets of size k in G and checking if every connected component of $G - X$ is in some graph class Π_i for $i \in [d]$. This gives us an algorithm with running time $\binom{h(k)}{k} h(k)^{\mathcal{O}(1)}$ where $h(k) = 2(s(k))^2 + k$.

Let us now look at the case where there is a component C of $G - X$ of size more than $s(k)$. Let Π_j be the graph class which C belongs to. Let $R = V \setminus (X \cup C)$. Since X is a separator

of size at most k with separation (C, R) , we can conclude that $|R| \leq s(k)$. Hence we can conclude that $X \cup R$ is a modulator of size at most $s(k) + k$ such that $G[C] = G - (X \cup R)$ is a graph in graph class Π_j . Hence we can conclude that G has a modulator of size at most $g(k) = s(k) + k$ to the graph class Π_j .

Our algorithm first guesses the graph class Π_j and then uses the FPT algorithm for Π_j -Vertex Deletion to find a modulator S of size $g(k)$ such that $G - S$ is in the graph class Π_j .

We know that (C, X, R) is a partition of $V(G)$. Let (S_{CX}, S_R) be the partition of S where $S_{CX} = S \cap (C \cup X)$ and $S_R = S \cap R$. The algorithm goes over all 2-partitions of S to guess the partition (S_{CX}, S_R) .

Claim 3.1.1. *For every component Q in the graph $G - X$ such that Q is not in the graph class Π_j , we have $S_R \cap V(Q) \neq \emptyset$.*

Proof. Suppose $S_R \cap V(Q) = \emptyset$. Since Q is not in the graph class Π_j and $G[C]$ is in the graph class Π_j , we have $V(Q) \subseteq R$. Hence if $S_R \cap V(Q) = \emptyset$, we have $S \cap V(Q) = \emptyset$. But then this contradicts the fact that $G - S$ is in the graph class Π_j as Q is not in the graph class Π_j and Π_j is a hereditary graph class. \square

For every vertex $v \in S_R$, let Q_v denote the component in $G - X$ that contains v . Note that the neighborhood of Q_v in the graph G is a subset of X which is of size at most k . Since we know that $Q_v \cup N(Q_v)$ is contained in the set $X \cup R$, we can conclude that (G, k) is a YES-instance if and only if $(G - (Q_v \cup N(Q_v)), k - |N(Q_v)|)$ is a YES-instance.

The following proposition bounds the number of small connected vertex subsets with a small neighborhood. This helps us to guess the subset Q_v .

Proposition 3.1.1. ([74], [94]) *Let $G = (V, E)$ be a graph. For every $v \in V$, and integers $b, f \geq 0$, the number of connected vertex subsets $B \subseteq V$ such that*

- (a) $v \in B$,

(b) $|B| \leq b + 1$ and

(c) $|N(B)| \leq f$

is at most $\binom{b+f}{b}$ and can be enumerated in time $O(n \cdot b^2 \cdot f \cdot (b+f) \cdot \binom{b+f}{b})$ by making use of polynomial space.

We have the following branching rule.

Branching Rule 1. Let $v \in S_R$. Using the enumeration algorithm from Proposition 3.1.1, go over all connected vertex subsets $B \subseteq V$ such that $v \in B$, $|B| \leq b + 1$, and $|N(B)| \leq f$ where $b = s(k)$ and $f = k$ and return the instance $(G - B, k - |N(B)|)$.

The branching rule is safe because in one of the branches, the algorithm rightfully guesses $B = Q_v$. The algorithm repeats the branching rule for all vertices $v \in S_R$. Hence we can assume that the current instance is such that $S_R = \emptyset$. We update the sets X and R by accordingly deleting the removed vertices. Let (G', k') be the resulting instance. We have the following claim.

Claim 3.1.2. The set X is such that $|X| \leq k'$ and $G' - X$ is in the graph class Π_j .

The proof of the claim comes from the observation that as $S \cap R = \emptyset$, every component other than C does not intersect with S . Hence from Claim 3.1.1, these components have to be in the graph class Π_j as $G - S$ is in the graph class Π_j .

The algorithm now again uses the FPT algorithm for the graph class Π_j to obtain the solution of size k' thereby solving the problem.

We summarize the algorithm below for the case when there is exactly one component in $G - X$ of size more than $s(k)$.

1. For any of the given graph classes check whether the given graph G has a modulator of size at most $g(k)$. If none of them has such a set, then return NO. Otherwise, we do Steps 2 and 3 for all graph classes Π_j having a modulator S of size at most $g(k)$.

2. Go over all 2-partitions (S_{CX}, S_R) of S . For each $v \in S_R$, apply Branching Rule 1. Let (G', k') be the resulting instance.
3. Check whether the graph G' has a Π_j -deletion set of size at most k' . If yes, return YES. Else return NO.

Running Time: Let $f(k)n^{O(1)}$ be the maximum over all the running times of Π_j VERTEX DELETION for $j \in [d]$. We use $d \cdot f(g(k)) \cdot 2^{g(k)}n^{O(1)}$ time to obtain set S and its 2-partition where $g(k) = s(k) + k$. We use overall $O(n(g(k) + 1)^{k+1})$ time to enumerate the connected vertex sets in Branching Rule 1. The branching factor of the rule is bounded by $(g(k) + 1)^{k+1}$ and the depth is bounded by k . Since the choices of $v \in S_R$ is bounded by $g(k)$, exhaustive application of Branching Rule takes at most $g(k)^{k(k+2)}n^{O(1)}$ time. Finally we apply the algorithm for Π_j -VERTEX DELETION again taking at most $f(k)n^{O(1)}$ time.

Hence the overall running time is bounded by $d \cdot f(g(k)) \cdot 2^{g(k)}(g(k) + 1)^{k(k+2)}n^{O(1)}$. \square

The proof of Theorem 4 that the problem is FPT in general graphs follows from Theorem 6 and Theorem 7.

3.2 Deletion to scattered classes with finite forbidden families

Unfortunately, the algorithm for INDIVIDUALLY TRACTABLE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ VERTEX DELETION in Theorem 4 has two drawbacks. One is that the algorithm is only a non-uniform FPT algorithm. The other is that the algorithm has a huge running time due to the gargantuan overhead from applying Theorem 6.

We now look into a special case of INDIVIDUALLY TRACTABLE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ VERTEX DELETION where every graph class Π_i with $i \in [d]$ can be characterized by a finite forbidden family. Recall that Π_i VERTEX DELETION is FPT for each $i \in [d]$ from the

simple branching algorithm over vertices of the induced subgraphs H of the input graph G that is isomorphic to members of the finite forbidden family \mathcal{F}_i . Also, the properties that “graph G is in Π_i ” can be expressed in CMSO logic as we can hard code the graphs of \mathcal{F}_i in the formula. Hence the problem is indeed a special case of INDIVIDUALLY TRACTABLE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ VERTEX DELETION. In this section, we give an FPT algorithm for this case with running time much better when compared to that in Theorem 4.

Brief Outline of the section:

In Section 3.2.1, we first use the standard technique of iterative compression to obtain a tuple (G, k, W) of the input instance DISJOINT FINITE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -VDC where W is a $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator of size at most $k + 1$ and the aim is to obtain a solution of size at most k disjoint from W . We also add a requirement to the problem that some of the vertices are forced to be not in the solution which will be useful later.

In Subsection 3.2.2, we give an FPT algorithm for DISJOINT FINITE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -VDC in the special case when the solution that we are looking for leaves W in a single component. The algorithm uses the standard technique of important separators [126] where we branch on vertices of a finite forbidden set in the graph plus some important separators corresponding to the set.

Finally in Subsection 3.2.3, we handle general instances of DISJOINT FINITE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -VDC. We focus on instances where the solution separates W . We guess $W_1 \subset W$ as the part of W that occurs in some single connected component after deleting the solution. The algorithm finds a set \mathcal{R} of $2^{poly(k)}$ vertices one of which intersects the solution and do a branching on vertices of \mathcal{R} . Finding \mathcal{R} involves a recursive subprocedure described below.

Since, the solution separates W , we know that it contains a $W_1 - (W \setminus W_1)$ separator X . It can be proven that X is a ‘special’ kind of important separator (its definition is tailored to our problem). The algorithm uses the technique of tight separator sequences [119].

It guesses the integer ℓ which is the size of the part of the solution present in the graph containing W_1 after removing X . The algorithm then constructs the tight separator sequence corresponding to ℓ and finds the separator P furthest from W_1 in the sequence such that there is a $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator of size ℓ in the graph containing W_1 after removing P . If separator X either intersects P or dominates the other, then we easily find vertices that intersects the solution and thereby in \mathcal{R} . The harder case is when the two separators are incomparable. Let Y be the set of vertices Y that is reachable from W_1 after deleting P . The algorithm constructs a graph gadget of $k^{O(1)}$ vertices whose appropriate attachment to the boundary P of the graph $G[Y]$ gives a graph G' which preserves the part of the solution of G present in $G[Y]$. Since this part of the solution is strictly smaller in size, the algorithm can find the set of vertices hitting the solution for G by recursively finding a similar set in G' and adding it to \mathcal{R} .

3.2.1 Iterative Compression

We use the standard technique of iterative compression to transform the FINITE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ VERTEX DELETION problem into the following problem DISJOINT FINITE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ VERTEX DELETION COMPRESSION (DISJOINT FINITE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -VDC for short). Standard ideas from iterative compression shows that an FPT algorithm with running time $\mathcal{O}^*(f(k))$ for the latter gives a $\mathcal{O}^*(2^{k+1}f(k))$ time algorithm for the former. We refer to [45] for the details.

DISJOINT FINITE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -VDC

Input: A graph G , an integer k , finite forbidden sets $\mathcal{F}_1, \mathcal{F}_2, \dots, \mathcal{F}_d$ for graph classes $\Pi_1, \Pi_2, \dots, \Pi_d$ and a subset W of $V(G)$ such that W is a $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator of size $k+1$.

Parameter: k

Question: Is there a subset $Z \subseteq V(G) \setminus W, |Z| \leq k$ such that Z is a $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator of the graph G ?

We now define an extension of DISJOINT FINITE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -VDC to incorporate the notion of undeletable vertices. The input additionally contains a set $U \subseteq V(G)$ of undeletable vertices and we require the solution $Z \subseteq V(G)$ to be disjoint from U .

DISJOINT FINITE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -VDC WITH UNDELETABLE VERTICES

Input: A graph G , an integer k , finite forbidden sets $\mathcal{F}_1, \mathcal{F}_2, \dots, \mathcal{F}_d$ for graph classes $\Pi_1, \Pi_2, \dots, \Pi_d$ a subset W of $V(G)$ such that W is a $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator of size $k + 1$ and a subset $U \subseteq V(G)$.

Parameter: k

Question: Is there a subset $Z \subseteq V(G) \setminus (W \cup U), |Z| \leq k$ such that Z is a $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator of the graph G ?

We have the following reduction rule which allows us to remove components in the graph belonging to some graph class Π_i for $i \in [d]$.

Reduction Rule 1. *If a connected component of G belongs to some graph class Π_i for $i \in [d]$, then remove all the vertices of this connected component.*

Lemma 3.2.1. *Reduction Rule 1 is safe.*

Proof. Let \mathcal{X} be the connected component of G removed to get an instance (G', k, W') . We claim that (G, k, W) is a YES-instance if and only if (G', k, W') is also a YES-instance. Let Z be a solution of G of size at most k . Since G' is an induced subgraph of G , Z is also a solution of G' as well. Conversely, suppose Z' is the solution of size k for graph G' . Then every connected component of the graph $G' \setminus Z'$ belongs to some graph class Π_i for $i \in [d]$. Since \mathcal{X} also belongs to some graph class Π_i for some $i \in [d]$, we have that Z' is also a solution for the graph G . □

We now develop the notion of forbidden sets which can be used to identify if a connected component of a graph belongs to any of the classes Π_i for $i \in [d]$.

Definition 3.2.1. *We say that a subset of vertices $C \subseteq V(G)$ is a **forbidden set** of G if C*

occurs in a connected component of G and there exists a subset $C_i \subseteq C$ such that $G[C_i] \in \mathcal{F}_i$ for all $i \in [d]$ and C is a minimal such set.

Clearly, if a connected component of G contains a forbidden set, then it does not belong to any of the graph classes Π_i for $i \in [d]$. We note that even though the forbidden set C is of finite size, the lemma below rules out the possibility of a simple algorithm involving just branching over all the vertices of C .

Lemma 3.2.2. *Let G be a graph and $C \subseteq V(G)$ be a forbidden set of G . Let Z be a $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator of G . Then Z disconnects C or $Z \cap C \neq \emptyset$.*

Proof. Suppose Z is disjoint from C . We know that C cannot occur in a connected component \mathcal{X} of $G \setminus Z$ as \mathcal{X} cannot belong to any graph class Π_i for $i \in [d]$ due to the presence of subsets $C_i \subseteq C$ such that $G[C_i] \in \mathcal{F}_i$. Hence Z disconnects C . \square

3.2.2 Finding non-separating solutions

In this section, we focus on solving instances of DISJOINT FINITE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -VDC WITH UNDELETABLE VERTICES which have a non-separating property defined as follows.

Definition 3.2.2. *Let (G, k, W) be an instance of DISJOINT FINITE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -VDC WITH UNDELETABLE VERTICES and Z be a solution for this instance. Then Z is called a **non-separating** solution if W is contained in a single connected component of $G \setminus Z$ and **separating** otherwise. If an instance has only separating solutions, we call it a separating instance. Otherwise, we call it non-separating.*

We now describe the following lemma on important separators which is helpful in our algorithm to compute non-separating solutions with undeletable vertices. It was originally stated without the condition that the separators are disjoint from U . But what is stated follows from the observation that by replacing each vertex $u \in U$ with $k + 1$ copies of u

that forms a clique, every separator in the new graph of size at most k are those in the original graph which are disjoint from U .

Lemma 3.2.3. [37] *For every $k \geq 0$ and subsets $X, Y, U \subseteq V(G)$, there are at most 4^k important $X - Y$ separators of size at most k disjoint from U . Furthermore, there is an algorithm that runs in $\mathcal{O}(4^k kn)$ time that enumerates all such important $X - Y$ separators and there is an algorithm that runs in $n^{\mathcal{O}(1)}$ time that outputs one arbitrary component-maximal $X - Y$ separator disjoint from U .*

We now have the following lemma which connects the notion of important separators with non-separating solutions to our problem.

Lemma 3.2.4. *Let (G, k, W, U) be an instance of DISJOINT FINITE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -VDC WITH UNDELETABLE VERTICES obtained after exhaustively applying Reduction Rule 1 and Z be a non-separating solution. Let v be a vertex such that Z is a $\{v\} - W$ separator. Then there is a solution Z' which contains an important $\{v\} - W$ separator of size at most k in G and disjoint from U .*

Proof. Since we have applied Reduction Rule 1 as long as it is applicable, there is no connected component \mathcal{X} of G that is disjoint from W . Hence every component of G , in particular the component containing v intersects with W . Therefore, since the solution Z disconnects v from W , it must contain a minimal non-empty $\{v\} - W$ separator A which is disjoint from U . If A is an important $\{v\} - W$ separator, we are done. Else there is an important $\{v\} - W$ separator B dominating A which is also disjoint from U . We claim that $Z' = (Z \setminus A) \cup B$ is also a solution. Clearly $|Z'| \leq |Z|$. Suppose that there exists a forbidden set C in the graph $G \setminus Z'$. Let \mathcal{X} be the connected component of $G \setminus Z'$ containing C . Suppose \mathcal{X} is disjoint from W . Then there exists a connected component \mathcal{Y} of $G \setminus W$ containing \mathcal{X} , contradicting that W is a $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator. Hence \mathcal{X} must intersect W . Since $B \subseteq Z'$ disconnects v from W , we can conclude that \mathcal{X} is not contained in $R_G(v, B)$ as if so it cannot intersect with W .

By the definition of Z' , any component of the graph $G \setminus Z'$ which intersects $Z \setminus Z' = A \setminus B$ has to be contained in the set $R_G(v, B)$. Hence the component \mathcal{X} is disjoint from $Z \setminus Z'$. Thus, there exists a component \mathcal{H} of the graph $G \setminus Z$ containing \mathcal{X} . But this contradicts that Z is a $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator. \square

We use the above lemma along with Lemma 3.2.3 to obtain our algorithm for non-separating instances. The algorithm finds a minimal forbidden set C in polynomial time which by definition is of bounded size. Then it branches on the set C and also on $\{v\} - W$ important separators of size at most k of G for all $v \in C$.

Lemma 3.2.5. *Let (G, k, W, U) be a non-separating instance of DISJOINT FINITE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -VDC WITH UNDELETABLE VERTICES. Then the problem can be solved in $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ time.*

Proof. We first apply Reduction Rule 1 exhaustively. If the graph is empty, we return YES. Else, there is a connected component of G which does not belong to any graph classes Π_i for $i \in [d]$. Therefore, there exists a forbidden set $C \subseteq V(G)$ of G present in this connected component. We find C as follows. We check for each graph class Π_i , if a graph in \mathcal{F}_i exists as an induced subgraph for a particular connected component \mathcal{X} of G . If so, we take the union of the vertices of these induced graphs. We then make the set minimal by repeating the process of removing a vertex and seeing if the set remains a forbidden set.

We branch in $|C \setminus (W \cup U)|$ -many ways by going over all the vertices $v \in C \setminus (W \cup U)$ and in each branch, recurse on the instance $(G - v, k - 1, W, U)$. Then for all $v \in C$, we branch over all important $\{v\} - W$ separators X of size at most k in G disjoint from U and recurse on instances $(G \setminus X, k - |X|, W, U)$.

We now prove the correctness of the algorithm. Let $Z \subseteq V(G) \setminus (W \cup U)$ be a solution of the instance. From Lemma 3.2.2, we know that a forbidden set C of G is disconnected by Z or $Z \cap C \neq \emptyset$. In the latter case, we know that Z contains a vertex $x \in C \setminus (W \cup U)$ giving us one of the branched instances obtained by adding x into the solution.

Now we are in the case where C is disconnected by Z . Since Reduction rule 1 is applied exhaustively, the connected component containing C also contains some vertices in W . Since Z is a non-separating solution, W goes to exactly one connected component of $G \setminus Z$ and there exists some non-empty part of C that is not in this component. Hence, there exists some vertex $x \in C$ that gets disconnected from W by Z . From Lemma 3.2.4, we know that there is also a solution Z' which contains an important $\{x\} - W$ separator of size at most k in G disjoint from U . Since we have branched over all such $\{x\} - W$ important separators disjoint from U , we have correctly guessed on one such branch.

We now bound the running time. Let p be the size of the maximum sized graph present among all families \mathcal{F}_i . We then have $|C| \leq pd$ and any forbidden set in G can be obtained via brute force in n^{pd} time. For each $i \in [k]$, we know that there are at most 4^i important separators of size $1 \leq i \leq k$ disjoint from U which can be enumerated using Lemma 3.2.3 in $\mathcal{O}(4^i \cdot i \cdot n)$ time. For the instance (G, k, W) , if we branch on $v \in C$, k drops by 1 and if we branch on a $\{v\} - W$ separator of size i , k drops by i . Hence if $T(k)$ denotes the time taken for the instance (G, k, W) , we get the recurrence relation $T(k) = pdT(k-1) + \sum_{i=1}^k 4^i T(k-i)$. Solving the recurrence taking into account that p and d are constants, we get that $T(k) = 2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$. \square

3.2.3 Solving general instances

We now solve general instances of DISJOINT FINITE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -VDC WITH UNDELETABLE VERTICES using the algorithm for solving non-separating instances as a subroutine. Hence we focus on solving separating instances of DISJOINT FINITE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -VDC WITH UNDELETABLE VERTICES. We guess a subset $W_1 \subset W$ such that for a solution Z , W_1 is exactly the intersection of W with a connected component of $G \setminus Z$. For $W_2 = W \setminus W_1$, we are looking for a solution Z containing a $W_1 - W_2$ separator. Formally, let $W = W_1 \uplus W_2$ be a set of size $k+1$ which is a $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator. We look for a set $Z \subseteq V(G) \setminus (W \cup U)$ of size at most k such that Z is a $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -

modulator, Z contains a minimal (W_1, W_2) -separator X disjoint from U and W_1 occurs in a connected component of $G \setminus Z$.

From here on, we assume that the separating instance (G, k, W, U) of DISJOINT FINITE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -VDC WITH UNDELETABLE VERTICES is represented as (G, k, W_1, W_2, U) where $W = W_1 \uplus W_2$. We branch over all partitions of W into W_1 and W_2 which adds a factor of 2^{k+1} to the running time.

3.2.3.1 Disconnected case

We first focus on the particular case when the input instance is such that W_1 and W_2 are already disconnected in the graph G . We have the following lemma that allows us to focus on finding a non-separating solution in the connected component containing W_1 to reduce the problem instance.

Lemma 3.2.6. *Let $\mathcal{I} = (G, k, W_1, W_2, U)$ be an instance of DISJOINT FINITE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -VDC WITH UNDELETABLE VERTICES where W_1 and W_2 are in distinct components of G . Let Z be its solution such that W_1 exactly occurs in a connected component of $G \setminus Z$. Also let $R(W_1)$ be the set of vertices reachable from W_1 in G . Let $Z' = Z \cap R(W_1)$. Then $(G[R(W_1)], |Z'|, W_1, U \cap R(W_1))$ is a non-separating YES-instance of DISJOINT FINITE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -VDC WITH UNDELETABLE VERTICES and conversely for any non-separating solution Z'' for $(G[R(W_1)], |Z'|, W_1, U \cap R(W_1))$, the set $\hat{Z} = (Z \setminus Z') \cup Z''$ is a solution for the original instance such that W_1 exactly occurs in a connected component of $G \setminus \hat{Z}$.*

Proof. Suppose Z' is not a $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator for the graph G' . Then some component of $G' \setminus Z'$ contains a forbidden set C . The sets Z' and $Z \setminus Z'$ are disjoint as W_1 and W_2 are disconnected in G . Hence C is also in a connected component of $G \setminus Z$ giving a contradiction. Hence Z' is a solution for the instance $(G', |Z'|, W_1)$. Since the solution Z is such that W_1 is contained in a connected component of Z and $Z \setminus Z'$ is disconnected from

from Z' , Z' is a non-separating solution.

Conversely, suppose \hat{Z} is not a solution for the graph G . Then there exists a forbidden C in a connected component of $G \setminus \hat{Z}$. Either C is contained in the set $R(W_1)$ or in the set $NR(W_1) = V(G) \setminus R(W_1)$. If $C \subseteq R(W_1)$, C is also present in a connected component of the graph $G' \setminus Z''$ giving a contradiction that Z'' is a $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator of G' . If $C \subseteq NR(W_1)$, then C is contained in some connected component of the graph $G[NR(W_1)] \setminus (Z \setminus Z')$. Since Z' is disjoint from C , we conclude that C is a forbidden set in the graph $G \setminus (Z' \cup (Z \setminus Z')) = G \setminus Z$, giving a contradiction. \square

The following reduction rule allows us to use the algorithm for non-separating instance in the connected component of G containing W_1 .

Reduction Rule 2. *Let $\mathcal{I} = (G, k, W_1, W_2, U)$ be an instance of DISJOINT FINITE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -VDC WITH UNDELETABLE VERTICES where W_1 and W_2 are disconnected in G . Compute a non-separating solution Z' for the instance (G', k', W_1, U') where $G' = G[R(W_1)]$, $U' = U \cap R(W_1)$ and k' is the least integer $i \leq k$ for which (G', i, W_1, U') is a YES-instance. Delete Z' and return the instance $(G \setminus Z', k - |Z'|, W_2, U)$.*

The safeness of Reduction Rule 2 follows from Lemma 3.2.6. The running time for the reduction is $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ which comes from that of the algorithm in Lemma 3.2.5.

We now introduce the notion of good separators, tight separator sequences and t -boundaried graphs which are used to design the algorithm.

3.2.3.2 Good Separators and Tight Separator Sequences

We first look at a type of $W_1 - W_2$ separators such that the graph induced on the vertices reachable from W_1 after removing the separator satisfies the property as defined below.

Definition 3.2.3. *Let (G, k, W_1, W_2, U) be an instance of DISJOINT FINITE*

$(\Pi_1, \Pi_2, \dots, \Pi_d)$ -VDC WITH UNDELETABLE VERTICES. For integer ℓ , we call a $W_1 - W_2$ separator X in G (ℓ, U) -**good** if there exists a set K of size at most ℓ such that $K \cup X$ is a $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator for the graph $G[R[W_1, X]]$ with $(K \cup X) \cap U = \emptyset$. Else we call it (ℓ, U) -**bad**. If $U = \emptyset$, we call it ℓ -good and ℓ -bad respectively.

We now show that (ℓ, U) -good separators satisfy a monotone property if we compare them using their reachability sets.

Lemma 3.2.7. *Let (G, k, W_1, W_2, U) be an instance of DISJOINT FINITE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -VDC WITH UNDELETABLE VERTICES and let X and Y be disjoint $W_1 - W_2$ separators in G such that X covers Y and $(X \cup Y) \cap U = \emptyset$. If the set X is (ℓ, U) -good, then Y is also (ℓ, U) -good.*

Proof. Let us define graphs $G_X = G[R[W_1, X]]$ and $G_Y = G[R[W_1, Y]]$. Let K be a subset of size at most ℓ such that $K \cup X$ is a $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator for the graph G_X with $(K \cup X) \cap U = \emptyset$. Let $K' = K \cap R[W_1, Y]$. Note that since $K' \subseteq K$, we have $K' \cap U = \emptyset$. We claim that $K' \cup Y$ is a $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator for the graph G_Y proving that Y is (ℓ, U) -good.

Suppose $K' \cup Y$ is not a $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator. Then there exists a forbidden set $C \subseteq R[W_1, Y]$ contained in a single component of $G_Y \setminus (K' \cup Y)$. Since $C \subseteq R[W_1, Y] \subseteq R[W_1, X]$ and X and Y are disjoint, C does not intersect X . Also C does not contain any vertices in $K \setminus K'$ as Y disconnects the set from C . Hence C is disjoint from $K \cup X$. Since C lies in a single connected component of $G_Y \setminus (K' \cup Y)$ we can conclude that C occurs in a single connected component of the graph $G_X \setminus (K \cup X)$ giving a contradiction that X is (ℓ, U) -good. \square

We now define the notion of (ℓ, U) -important separators similar to important separators.

Definition 3.2.4. *Let (G, k, W_1, W_2, U) be an instance of DISJOINT FINITE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -VDC WITH UNDELETABLE VERTICES and let X and Y be $W_1 - W_2$*

separators in G such that Y dominates X and $(X \cup Y) \cap U = \emptyset$. Let ℓ be the smallest integer i for which X is (i, U) -good. If Y is (ℓ, U) -good, then we say that Y **well-dominates** X . If X is (ℓ, U) -good and there is no $Y \neq X$ which well-dominates X , then we call X as **(ℓ, U) -important**.

The following lemma allows us to assume that the solution of the instance (G, k, W_1, W_2, U) contains an (ℓ, U) -important $W_1 - W_2$ separator for some appropriate value of ℓ .

Lemma 3.2.8. *Let (G, k, W_1, W_2, U) be an instance of DISJOINT FINITE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -VDC WITH UNDELETABLE VERTICES and Z be a solution. Let $P \subseteq Z$ be a non-empty minimal $W_1 - W_2$ separator in G and let P' be a $W_1 - W_2$ separator in G well-dominating P . Then there is also a solution Z' for the instance containing P' .*

Proof. Let $Q = Z \cap R[W_1, P]$. Note that Q is a $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator for the graph $G[R[W_1, P]]$ with $Q \cap U = \emptyset$. Let $Q' \supseteq P'$ be a smallest $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator for the graph $G[R[W_1, P']]$ extending P' with $Q' \cap U = \emptyset$. We claim that $Z' = (Z \setminus Q) \cup Q'$ is a solution for the instance (G, k, W_1, W_2, U) . Since P' well-dominates P , $|Z'| \leq |Z|$ and $Z' \cap U = \emptyset$. Also note that $Z' \cap U = \emptyset$. We now show that Z' is a $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator. Suppose not. Then there exists a forbidden subset C present in a connected component \mathcal{X} of $G \setminus Z'$.

We first consider the case when \mathcal{X} is disjoint from the set $Z \setminus Z'$. Then there is a component \mathcal{H} in $G \setminus Z$ which contains \mathcal{X} and hence C , contradicting that Z is a solution. We now consider the case when \mathcal{X} intersects $Z \setminus Z'$. By definition of Z' , \mathcal{X} is contained in the set $R(W_1, P')$. Since $Z' \setminus Q'$ is disjoint from $R(W_1, P')$ and is separated from $R(W_1, P')$ by just P' , we can conclude that \mathcal{X} and hence C is contained in a single connected component of $G[R[W_1, P']] \setminus Q'$. But this contradicts that Q' is a $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator in the graph $G[R[W_1, P']]$. \square

We now define the notion of a tight separator sequence. It gives a natural way to partition the graph into parts with small *boundaries*.

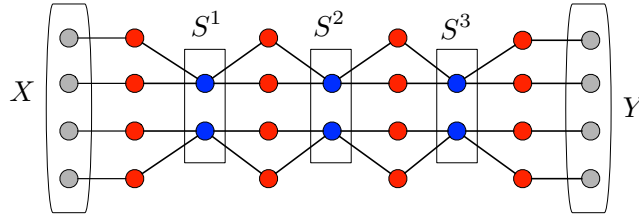


Figure 3.1: An $X - Y$ tight separator sequence of order two with $U = \emptyset$.

Definition 3.2.5. An $X - Y$ tight separator sequence of order k with undeletable set U of a graph G with $X, Y, U \subseteq V(G)$ is a set \mathcal{H} of $X - Y$ separators such that

- every separator has size at most k ,
- the separators are pairwise disjoint,
- every separator is disjoint from U ,
- for any pair of separators in the set, one covers another and
- the set is maximal with respect to the above properties.

See Figure 3.1 for an example of a tight separator sequence.

Lemma 3.2.9. Given a graph G , disjoint vertex sets X, Y and integer k , a tight separator sequence \mathcal{H} of order k with undeletable set U can be computed in $|V(G)|^{\mathcal{O}(1)}$ time.

Proof. Similar to what we mentioned while stating Lemma 3.2.3, we first replace every vertex $u \in U$ in our graph G with $k + 1$ copies of u forming a clique. Note that every separator in the new graph of size at most k are those in the original graph which are disjoint from U .

In this new graph, we check using the minimum cut algorithm if there is an $X - Y$ separator of size at most k . If not, we stop the procedure. Else we compute an arbitrary component-maximal $X - Y$ separator S of size at most k using the polynomial time algorithm in Lemma 3.2.3. We add S to the family \mathcal{H} , set Y to S , and repeat the process. We claim that \mathcal{H} is a

tight separator sequence of order k with an undeletable set U after the procedure terminates. It is clear that the first four properties of tight separator sequence are satisfied by \mathcal{H} in any iteration. Suppose \mathcal{H} is not maximal and hence an $X - Y$ separator P disjoint from U can be added. If P covers one of the separators S' in \mathcal{H} , it contradicts the component-maximality of S' at the time it was added to \mathcal{H} . Else P is covered by all the separators in \mathcal{H} which contradicts the termination of the procedure after the last separator in \mathcal{H} was added. This completes the proof. \square

In the proof, it can be seen that the separators S in \mathcal{H} can be totally ordered by the subset relation of the reachability sets $R(X, S)$. Hence \mathcal{H} is rather called a sequence than a family of separators.

3.2.3.3 Boundaried graphs

We now define the notion of boundaried graph where we associate a subset of its vertices as its boundary. We can then construct graphs by gluing two such graphs along the boundary.

Definition 3.2.6. *A t -boundaried graph G is a graph with t distinguished labelled vertices. We call the set of labelled vertices $\partial(G)$ the boundary of G and the vertices in $\partial(G)$ terminals. Let G_1 and G_2 be two t -boundaried graphs with the graphs $G_1[\partial(G_1)]$ and $G_2[\partial(G_2)]$ being isomorphic. Let $\mu : \partial(G_1) \rightarrow \partial(G_2)$ be a bijection which is an isomorphism of the graphs $G_1[\partial(G_1)]$ and $G_2[\partial(G_2)]$. We denote the graph $G_1 \otimes_{\mu} G_2$ as a t -boundaried graph obtained by the following gluing operation. We take the union of graphs G_1 and G_2 and identify each vertex $x \in \partial(G_1)$ with vertex $\mu(x) \in \partial(G_2)$. The t -boundary of the new graph is the set of vertices obtained by unifying.*

Definition 3.2.7. *A t -boundaried graph with an annotated set is a t -boundaried graph with a second set of distinguished but unlabelled vertices disjoint from the boundary. The set of annotated vertices is denoted by $\Delta(G)$.*

3.2.3.4 Algorithm

We design a recursive algorithm MAIN-ALGORITHM which takes as input the instance $\mathcal{I} = (G, k, W_1, W_2, U)$ and outputs YES if there exists a solution $Z \subseteq V \setminus (W_1 \cup W_2 \cup U)$ such that every connected component of $G - Z$ belongs to some graph class Π_i for $i \in [d]$.

Description of MAIN-ALGORITHM procedure: The MAIN-ALGORITHM procedure initially checks if Reduction Rule 1 is applicable for \mathcal{I} . Then it checks if $(G, k, W_1 \cup W_2, U)$ is a non-separating YES-instance using the algorithm from Lemma 3.2.5. If not, it checks if Reduction Rule 2 is applicable.

After these steps, we know that any solution Z of \mathcal{I} contains an (ℓ, U) -good $W_1 - W_2$ separator X in the graph G for some integer $0 \leq \ell \leq k$ with $|X| = \lambda > 0$. Using Lemma 3.2.8, we can further assume that the separator X is (ℓ, U) -important. Since $\lambda > 0$, we have $Z \cap R(W_1, X) \subset Z$ as X is not part of the set $Z \cap R(W_1, X)$. Hence $\ell = |Z \cap R(W_1, X)| < |Z| \leq k$. Hence we can conclude that $0 \leq \ell < k$ and $1 \leq \lambda \leq k$.

The MAIN-ALGORITHM procedure now calls a subroutine BRANCHING-SET with input as $(\mathcal{I}, \lambda, \ell)$ for all values $0 \leq \ell < k$ and $1 \leq \lambda \leq k$. The BRANCHING-SET subroutine returns a vertex subset $\mathcal{R} \subseteq V(G)$ of size $2^{\text{poly}(k)}$ such that for every solution $Z \subseteq (V(G) \setminus U)$ of the given instance \mathcal{I} containing an (ℓ, U) -important $W_1 - W_2$ separator X of size at most λ in G , the set \mathcal{R} intersects Z . The MAIN-ALGORITHM procedure then branches over all vertices $v \in \mathcal{R}$ and recursively run on the input $\mathcal{I}' = (G - v, k - 1, W_1, W_2, U)$.

Description of BRANCHING-SET procedure:

We first check if there is a $W_1 - W_2$ separator of size λ in the graph G with the vertices contained in the set $V \setminus U$. If there is no such separator, we declare the tuple invalid. Else we execute the algorithm in Lemma 3.2.9 to obtain a tight $W_1 - W_2$ separator sequence \mathcal{T} of order λ and undeletable set U .

Let $\mathcal{T} = O_1, O_2, \dots, O_q$ for some integer q . We partition \mathcal{T} into (ℓ, U) -good and (ℓ, U) -bad separators as follows. Recall Lemma 3.2.7 where we proved that if X and Y are disjoint $W_1 - W_2$ separators in G such that X covers Y and X is (ℓ, U) -good, then Y is also (ℓ, U) -good. From this we can conclude that the separators in the sequence \mathcal{T} are such that if they are neither all (ℓ, U) -good nor all (ℓ, U) -bad, there exist an $i \in [q]$ where O_1, \dots, O_i are (ℓ, U) -good and O_{i+1}, \dots, O_q are (ℓ, U) -bad. We can find i in $\lceil \log q \rceil$ steps via binary search if at each step, we know of a way to check if for a given integer $j \in [q-1]$ if O_j is (ℓ, U) -good and O_{j+1} is (ℓ, U) -bad. In the case $j = q$, we only check if O_j is (ℓ, U) -good and if so conclude that all the separators in the sequence are (ℓ, U) -good. In the case where $j = 0$, we only check if O_j is (ℓ, U) -bad and if so conclude that all the separators in the sequence are (ℓ, U) -bad.

In any case, we need a procedure to check whether a given separator P is (ℓ, U) -good or not. From the definition of (ℓ, U) -good separator, this translates to checking if there is a $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator of size at most ℓ in the graph $G[R(W_1, P)]$ such that the solution is disjoint from $W_1 \cup U$. Note that since P separates W_1 from W_2 , the set W_1 is a $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator in the graph $G[R(W_1, P)]$. Hence the problem translates to checking whether $\mathcal{I}_1 = (G[R(W_1, P)], \ell, W_1, U)$ is a YES-instance of DISJOINT FINITE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -VDC WITH UNDELETABLE VERTICES. This can be done by calling the MAIN-ALGORITHM procedure for the instance $(G[R(W_1, P)], \ell, W_1, U)$. Note that this is a recursive call in the initial MAIN-ALGORITHM procedure with \mathcal{I} as input where we called the BRANCHING-SET procedure with the associated solution size ℓ being strictly less than k .

If we do not find an integer i such that O_j is (ℓ, U) -good and O_{j+1} is (ℓ, U) -bad, or conclude that all the separators in the sequence are either (ℓ, U) -good or all are (ℓ, U) -bad, we declare that the tuple is not valid. Otherwise, we have a separator P_1 which is component maximal among all the good separators in \mathcal{T} if any exists, and separator P_2 which is component minimal among all the bad separators in \mathcal{T} if any exists. We initialize

the set $\mathcal{R} := P_1 \cup P_2$. For $i \in \{1, 2\}$, we do the following.

We go over every subset $P_i^r \subseteq P_i$. For each such subset, we compute a family \mathcal{H} of $|P_i^r|$ -boundaried graphs which consists of all graphs of size at most $k^{5(pd)^2}$ of which at most k are annotated. Note that the total number of such graphs is bounded by $2^{\binom{\gamma}{2}} \binom{\gamma}{k+1}$ for $\gamma = k^{5(pd)^2}$ and these can be enumerated in time $2^{\binom{\gamma}{2}} \binom{\gamma}{k+1} k^{\mathcal{O}(1)}$.

For every choice of $P_i^r \subseteq P_i$, for every annotated boundaried graph $\hat{G} \in \mathcal{H}$ with $|P_i^r|$ terminals and every possible bijection $\delta : \partial(\hat{G}) \rightarrow P_i^r$, we construct the glued graph $G_{P_i^r, \delta} = G[R[W_1, P_i]] \otimes_{\delta} \hat{G}$, where the boundary of $G[R[W_1, P_i]]$ is P_i^r . We then recursively call $\text{BRANCHING-SET}((G_{P_i^r, \delta} \setminus \tilde{S}, k - j, W_1, P_i \setminus P_i^r, U \cup V(\hat{G}) \setminus P_i^r), \lambda', \ell')$ for every $0 \leq \lambda' < \lambda$, $1 \leq j \leq k - 1$ and $0 \leq \ell' \leq \ell$, where \tilde{S} is the set of annotated vertices in \hat{G} . We add the union of all the vertices returned by these recursive instances to \mathcal{R} and return the resulting set.

This completes the description of the BRANCHING-SET procedure. We now proceed to the proof of correctness.

Correctness of MAIN-ALGORITHM and BRANCHING-SET procedure: We prove the correctness of MAIN-ALGORITHM by induction on k . The case when $k = 0$ is correct as we can check if \mathcal{I} is a YES-instance in polynomial time by checking if every connected component of G belongs to one of the graph classes Π_i for $i \in [d]$. We now move to the induction step with the induction hypothesis being that the MAIN-ALGORITHM procedure correctly runs for all instances \mathcal{I} where $k < \hat{k}$ for some $\hat{k} \geq 1$ and identifies whether \mathcal{I} is a YES-instance. We now look at the case when the algorithm runs on an instance with $k = \hat{k}$.

The correctness of the initial phase follows from the safeness of Reduction Rules 1, 2 and the correctness of the algorithm in the non-separating case. Let us now assume that the BRANCHING-SET procedure is correct. Hence the set \mathcal{R} returned by BRANCHING-SET procedure is such that it intersects a solution Z if it exists. Therefore $\mathcal{I} = (G, k, W_1, W_2, U)$

is a YES-instance if and only if $\mathcal{I}' = (G - v, k - 1, W_1, W_2, U)$ is a YES-instance for some $v \in \mathcal{R}$. Applying the induction hypothesis for MAIN-ALGORITHM with input instance \mathcal{I}' , we prove the correctness of MAIN-ALGORITHM.

It remains to prove the correctness of the BRANCHING-SET procedure. Note that all the calls of MAIN-ALGORITHM in this procedure have input instances checking for solutions strictly less than k . Hence these calls run correctly from the induction hypothesis when BRANCHING-SET is called in the MAIN-ALGORITHM procedure. Hence we only need to prove that BRANCHING-SET procedure is correct with the assumption that all the calls of MAIN-ALGORITHM in the procedure run correctly. We prove this by induction on λ . Recall that the sets P_1 and P_2 were identified via a binary search procedure described earlier using the calls of MAIN-ALGORITHM with values strictly less than k . Since we assume that the calls of MAIN-ALGORITHM runs correctly, the sets P_1 and P_2 were correctly identified if present.

We first consider the base case when $\lambda = 1$ where there is a $W_1 - W_2$ (ℓ, U) -good separator $X \subseteq Z$ of size one. Since X has size one, it cannot be incomparable with the separator P_1 . Hence the only possibilities are X is equal to P_1 , is covered by P_1 or covers P_1 . In the first case, we are correct as P_1 is contained in \mathcal{R} . The second case contradicts that X is (ℓ, U) -important $W_1 - W_2$ separator. We note that in the third case, we can conclude that X is covered by P_2 . This is because the other cases where X is equal to be P_2 or X covers P_2 cannot happen as P_2 is (ℓ, U) -bad and X is incomparable to P_2 cannot happen as both are of size one. Hence X covers P_1 and is covered by P_2 . But then X must be contained in the tight separator sequence \mathcal{T} contradicting that P_1 is component maximal. Hence the third case cannot happen.

We now move to the induction step with the induction hypothesis being that BRANCHING-SET procedure correctly runs for all tuples where $\lambda < \hat{\lambda}$ for some $\hat{\lambda} \geq 2$ and returns a vertex set that hits any solution for its input instance that contains an (ℓ, U) -important separator of size λ and not containing any vertices from U . We now look at the case when

the algorithm runs on a tuple with $\lambda = \hat{\lambda}$.

Let $Z \subseteq (V(G) \setminus U)$ be a solution for the instance \mathcal{I} containing an (ℓ, U) -important separator X . If X intersects $P_1 \cup P_2$ we are done as $\mathcal{R} \supseteq P_1 \cup P_2$ intersects X . Hence we assume that X is disjoint from $P_1 \cup P_2$. Suppose X is covered by P_1 . Then we can conclude that P_1 well-dominates X contradicting that X is (ℓ, U) -important $W_1 - W_2$ separator.

By Lemma 3.2.7, since X is (ℓ, U) -good and P_2 is not, X cannot cover P_2 . Suppose X covers P_1 and itself is covered by P_2 . Then X must be contained in the tight separator sequence \mathcal{T} contradicting that P_1 is component maximal. Hence this case also does not happen.

Incomparable Case:

Finally we are left with the case where X is incomparable with P_1 or with P_2 if P_1 does not exist. Without loss of generality, assume X is incomparable with P_1 . The argument in the case when P_1 does not exist follows by simply replacing P_1 with P_2 in the proof.

Let $K \subseteq Z$ be the $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator for the graph $G[R[W_1, X]]$ extending X , i.e. $X \subseteq K$. In other words, $K = Z \cap R[W_1, X]$. Since X is an (ℓ, U) -good separator of G , we have $|K \setminus X| \leq \ell$. If $P_1 \cap K$ is non-empty, we have that $P_1 \cap Z$ is non-empty. Since P_1 is contained in \mathcal{R} , the algorithm is correct as \mathcal{R} intersects Z . Hence we can assume that P_1 and K are disjoint.

Let $X^r = R(W_1, P_1) \cap X$ and $X^{nr} = X \setminus X^r$. Similarly, define $P_1^r = R(W_1, X) \cap P_1$ and $P_1^{nr} = P_1 \setminus P_1^r$. Since X and P_1 are incomparable, the sets X^r, X^{nr}, P_1^r and P_1^{nr} are all non-empty. Let $K^r = K \cap R[W_1, P_1]$ and $K^{nr} = K \setminus K^r$. Note that $X^r \subseteq K^r$ and $X^{nr} \subseteq K^{nr}$. See Figure 3.2.

We intend to show in the case when X and P_1 are incomparable, the set returned by one of the recursive calls of the BRANCHING-SET procedure hits the solution Z .

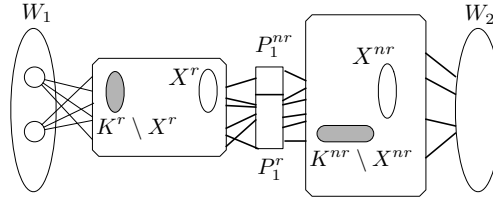


Figure 3.2: The case where X is incomparable with P_1

We now prove the following crucial lemma where we show that by carefully replacing parts outside of $R[W_1, P_1]$ with a small gadget, we can get a smaller graph G' such that K^r , the part of K inside the set $R[W_1, P_1]$ is an optimal $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator containing the $(|K^r \setminus X^r|, U')$ -important separator X^r in this graph for an appropriate subset $U' \subseteq V(G')$. This allows us to show that the instance of BRANCHING-SET corresponding to G' would return a set that intersects the optimal $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator K^r in G' and thereby the solution Z in G . Later, we will see that one of the recursive calls of BRANCHING-SET indeed correspond to G' .

Lemma 3.2.10. *Let $G_1 = G[R[W_1, P_1]]$ be a boundaried graph with P_1^r as the boundary. There exists a $|P_1^r|$ -boundaried graph \hat{G} which is at most $k^{5(pd)^2}$ in size with an annotated set of vertices $\Delta(\hat{G})$ of size at most k , and a bijection $\mu : \partial(\hat{G}) \rightarrow P_1^r$ such that the glued graph $G' = G_1 \otimes_{\mu} \hat{G}$ has the property that BRANCHING-SET procedure with input as $((G' \setminus \Delta(\hat{G}), |K^r|, W_1, P_1^{nr}, U \cup V(\hat{G}) \setminus P_1^r), |X^r|, |K^r \setminus X^r|)$ returns a set \mathcal{R}' that intersects K^r .*

Proof. To develop the intuitions behind the proof, we first prove that for the graph $G'' = G[R[W_1, X]]$, BRANCHING-SET with input as $((G'' \setminus K^{nr}, |K^r|, W_1, P_1^{nr}, U), |X^r|, |K^r \setminus X^r|)$ returns a set \mathcal{R}'' that intersects K^r .

Suppose \mathcal{R}'' does not intersect K^r . The set \mathcal{R}'' by definition intersects any $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator $Z'' \subseteq V(G'') \setminus U$ of size $|K^r|$ for the graph $G'' \setminus K^{nr}$ containing an $(|K^r \setminus X^r|, U)$ -important separator X'' . The set $K^r \subseteq V(G'') \setminus U$ is also a $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator of size $|K^r|$ for the graph $G'' \setminus K^{nr}$ and it contains a $W_1 - P_1^{nr}$ separator X^r . Hence, if we can

prove that the set X^r is a $(|K^r \setminus X^r|, U)$ -important separator in the graph $G'' \setminus K^{nr}$, we are done.

Suppose this is not the case. Then there exists a $(|K^r \setminus X^r|, U)$ -good separator $X'' \subseteq V(G'') \setminus U$ in the graph $G'' \setminus K^{nr}$ that well-dominates X^r . Note that since G'' is the graph $G[R[W_1, X]]$, the set of vertices reachable from W_1 after deleting X^r in the graph $G'' \setminus K^{nr}$ denoted by $R_{G'' \setminus K^{nr}}(W_1, X^r)$ is the set $R_{G \setminus K^{nr}}(W_1, X)$. If $X'' \neq X^r$, the set $R_{G \setminus K^{nr}}(W_1, X) = R_{G'' \setminus K^{nr}}(W_1, X^r) \subset R_{G'' \setminus K^{nr}}(W_1, X'')$ which cannot happen.

Note that the graph G'' can be viewed as the graph obtained by gluing two bounded graphs G_1 and G_2 both having boundary P_1^r where $G_1 = G[R[W_1, P_1]]$ and $G_2 = G[NR(W_1, P_1) \cap R(W_1, X) \cup P_1^r \cup K^{nr}]$ with the bijection being an identity mapping from P_1^r into itself. Unfortunately the graph G_2 is not of size $k^{\mathcal{O}(1)}$ size and hence does not satisfy the conditions required for the lemma. We now aim to construct a graph \hat{G} by keeping some $k^{\mathcal{O}(1)}$ vertices of G_2 .

Let $V_2 = (NR(W_1, P_1) \cap R(W_1, X)) \cup P_1^r \cup K^{nr}$. The set $V_2 \setminus (P_1^r \cup K^{nr})$ contains the vertices which are disconnected by P_1 from W_1 but are not disconnected from W_1 by X . We have $G_2 = G[V_2]$.

Marking Vertices of Forbidden Sets:

We now perform the following marking scheme on the graph G where we mark some vertices of V_2 to construct a smaller graph G' . Before this though, we need to define the following notations.

Let p denote the size of the maximum sized subgraph present among all the families \mathcal{F}_i . Let $\mathbb{H} = \{(H_1, \dots, H_d) : H_i \in \mathcal{F}_i, i \in [d]\}$ where each tuple (H_1, \dots, H_d) corresponds to a collection of graphs of a forbidden set. For $\mathcal{H} = (H_1, \dots, H_d) \in \mathbb{H}$, let $\mathbb{B}_{\mathcal{H}} = \{(B_1, \dots, B_d) : B_i \subseteq V(H_i), i \in [d]\}$. Let $\mathbb{P}_1^r = \{(Q_1, \dots, Q_d) : Q_i \subseteq P_1^r, |Q_i| \leq p, i \in [d]\}$.

Let $\mathbb{T}_{\mathcal{H}}$ be the collection of tuples (t_1, \dots, t_r) where t_i is a pair of elements $t_i^1, t_i^2 \in P_1^r \cup \{\emptyset\}$

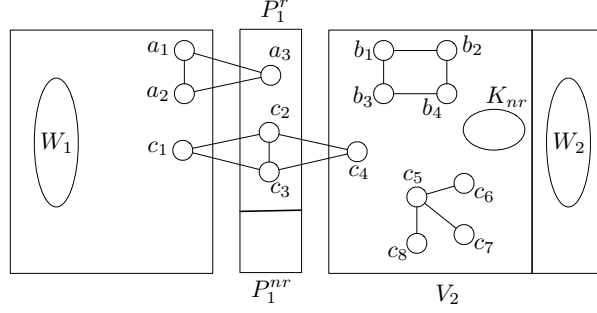


Figure 3.3: Example of a marked forbidden set

and $r = \left(\binom{\sum_{i \in [d]} |H_i|}{2}, i \in [r] \right)$. We use the bijection $\rho : \left(\binom{\cup_{i \in [d]} H_i}{2} \right) \rightarrow [r]$ so that $\rho(\{a, b\})$ denotes the index associated to the pair of vertices $a, b \in \cup_{i \in [d]} H_i$.

For all tuples $\langle \mathcal{H}, \mathcal{B}_{\mathcal{H}}, \mathcal{P}_1^r, \mathcal{T}_{\mathcal{H}} \rangle$ where $\mathcal{H} = (H_1, \dots, H_d) \in \mathbb{H}$, $\mathcal{B}_{\mathcal{H}} = (B_1, \dots, B_d) \in \mathbb{B}_{\mathcal{H}}$, $\mathcal{P}_1^r = (Q_1, \dots, Q_d) \in \mathbb{P}_1^r$ and $\mathcal{T}_{\mathcal{H}} = (t_1, \dots, t_r) \in \mathbb{T}_{\mathcal{H}}$, if there exists a forbidden set $C \subseteq (V_2 \cup V(G_1))$ of the graph $G \setminus K^{nr}$ such that

- For all $i \in [d]$, there exists a subset $C_i \subseteq C$ such that $G[C_i]$ is isomorphic to H_i ,
- for sets $C_i^+ = V_2 \cap C_i$, we have graphs $G[C_i^+]$ isomorphic to $H_i[B_i]$,
- the set $P_1^r \cap C_i = Q_i$ and
- for vertices $a_i \in C_i$ and $a_j \in C_j$ with $i, j \in [d]$, there is path P' from a_i to a_j in the graph $G \setminus K^{nr}$ such that the first and last vertex of P' in the set P_1^r that has a neighbor to the set $R(W_1, P_1)$ is $t_{\rho(a_i, a_j)}^1$ and $t_{\rho(a_i, a_j)}^2$ respectively, (When the path P' has only one such vertex v , we denote it by the pair $\{v, \emptyset\}$. If the path has no such vertex, then we denote it by the pair $\{\emptyset, \emptyset\}$, . Also note that the existence of such paths for all pair of vertices in C shows that $G[C]$ is connected in the graph $G \setminus K^{nr}$).

then for one such forbidden set C , we mark the set $C^+ = C \cap V_2$. Let M' be the set of vertices marked in this procedure. We call the corresponding forbidden sets C as marked forbidden sets.

See figure 3.3 for an example of a marked forbidden set. We have $\mathcal{H} = (H_1, H_2, H_3)$ where

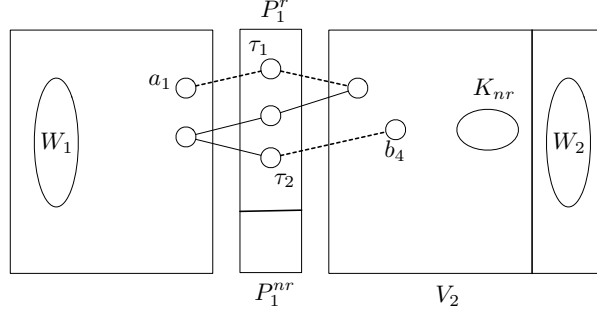


Figure 3.4: Example showing paths between vertices of a marked forbidden set

H_1 is a triangle, H_2 is a C_4 and H_3 has two connected components, one of which is K_4 after removal of an edge and the other is a claw $K_{1,3}$. The graph induced by the set of vertices $\{a_1, a_2, a_3\}$ is isomorphic to H_1 , the one induced by the set of vertices $\{b_1, b_2, b_3, b_4\}$ is isomorphic to H_2 and the one induced by the set of vertices $\{c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8\}$ is isomorphic to H_3 . We have $\mathcal{B}_{\mathcal{H}} = (B_1, B_2, B_3)$ where B_1 is a singleton vertex, B_2 is the cycle graph C_4 and B_3 has two connected components, one of which is a triangle and the other is a claw $K_{1,3}$. Notice that there are the graphs induced by H_1 , H_2 and H_3 when we restrict the set of vertices to V_2 . We have $\mathcal{P}'_1 = (Q_1, Q_2, Q_3)$ as $(\{a_3\}, \{\emptyset\}, \{c_2, c_3\})$.

See figure 3.4 where we look at a path between a_1 and b_4 of the same marked forbidden set. The first and last vertex of such a path is τ_1 and τ_2 respectively. Hence the entry of $\mathcal{T}_{\mathcal{H}}$ corresponding to the pair (a_1, b_4) is $\{\tau_1, \tau_2\}$.

We now bound the size of M' . We know that each graph class \mathcal{F}_i has a finite number of finite sized graphs. Let $f = \max_{i \in [d]} |\mathcal{F}_i|$. The size of \mathbb{H} is the number of tuples $\mathcal{H} = (H_1, \dots, H_d)$ which is at most f^d . Since $|H_i| \leq p$, the size of $\mathbb{B}_{\mathcal{H}}$ is bounded by the number of tuples (B_1, \dots, B_d) which is at most 2^{pd} . Since the set Q_i is of size at most p , the size of \mathbb{P}'_1 is bounded by $k^{(p+1)d}$. Each vertex in a pair in $\mathbb{T}_{\mathcal{H}}$ is a pair of vertices of P'_1 . The number of such pairs is bounded by $(k+1)^2$. Since $r = \binom{\sum_{i \in [d]} |H_i|}{2} \leq \binom{pd}{2}$, the size of $\mathbb{T}_{\mathcal{H}}$ is bounded by $((k+1)^2)^{\binom{pd}{2}}$. Overall, we can conclude that the number of tuples $\langle \mathcal{H}, \mathcal{B}_{\mathcal{H}}, \mathcal{P}'_1, \mathcal{T}_{\mathcal{H}} \rangle$ is at most $\eta = f^d 2^{pd} k^{(p+1)d} k^{2 \binom{pd}{2}}$. For each of these tuples we mark the set $C \cap V_2$ which is of size at most pd . Hence we can conclude that $|M'| \leq \eta pd$. The same bound holds for the vertices corresponding to the marked forbidden sets which we denote

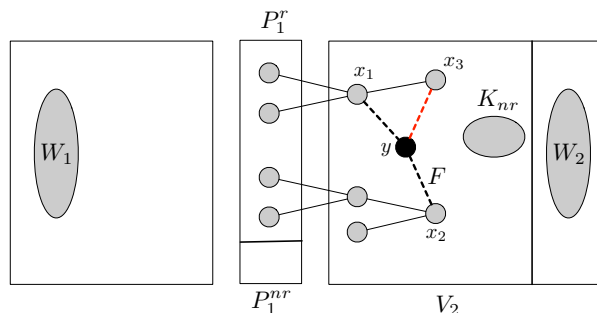


Figure 3.5: Forest F that provides required connectivities of marked forbidden set vertices. The vertices colored grey correspond to marked vertices and white correspond to other vertices of F . The forest F has a degree two path between x_1 and x_2 with all the internal vertices unmarked. If an unmarked vertex y in this path has an edge to some marked vertex x_3 , then the forest obtained by replacing an edge adjacent to y in the path with (x_3, y) also preserves the connectivities but has an unmarked vertex as a leaf giving a contradiction.

by M_F .

Preserving Connectivity of the Marked Forbidden Sets: We now aim to keep some vertices in V_2 other than those in M' so that for every marked forbidden set C , the graph $G[C]$ remains connected in the resulting graph. We also add the requirement that the connectivity between every vertex in C^+ and vertices in P_1^r and also between pairs of vertices in P_1^r in the graph $G[V_2]$ are preserved. Let F be the forest of minimum size in the graph G such that it satisfies these connectivity requirements. Note that $M' \subseteq V(F)$.

We now try to bound the size of the forest F . Note that any leaf of the forest F corresponds to some vertex in the marked forbidden set $M_F \cup P_1^r$. This is because it is not the case, then for some leaf vertex $u \in V(F)$, the forest $F - \{u\}$ also preserves the connectivities required for marked forbidden sets contradicting that F is the forest of minimum size. Hence the number of leaves is bounded by $\eta pd + k$.

By properties of any forest, the number of vertices of degree 3 or more is at most the number of leaves. Hence such vertices of F are also bounded by $\eta pd + k$. Hence it remains to bound the number of degree 2 vertices in F .

Let us focus on a degree 2 path P of F with endpoints either a leaf of F or degree at least

3 or any vertex in M_F or P_1^r . Suppose P has at least 3 internal vertices. We claim that all the internal vertices of P except the first and last internal vertices are not adjacent to any vertices of F in the graph G induced on $V(F)$ other than its neighbors in the path F . Suppose this is not the case for some internal vertex u with u_1 and u_2 being the two neighbors of u in P . Hence u is adjacent to some other vertex v of F . Note that the edge (u, v) is not in the forest F . Let us add this edge to the forest F creating a unique cycle C containing (u, v) . Without loss of generality, let u_1 be the other neighbor of u in C . Let F_1 be the forest created by adding the edge (u, v) and removing the edge (u, u_1) . Note that we now have a forest F_1 where u_1 is a leaf vertex that is not marked. Then $F_1 - \{u_1\}$ is also a forest that preserves the connectivities that F did with a fewer number of vertices. This contradicts that F is the forest with the minimum number of vertices. See Figure 3.5 for an illustration regarding this proof.

Since P does not have edges from the internal vertices to other vertices of F , we can contract these paths up to a certain length and preserve connectivities of F .

Let $G'_2 = G[V(F)]$. Let G_2 be the graph obtained from G'_2 by contracting all the degree 2 paths in the graph of length more than $4pd + 2$ to length $4pd + 2$. Since the number of degree 2 paths in F is bounded by $(2(|M_F| + |P_1|))^2$ and each path is bounded by size $4pd + 2$, we have $V(G_2) \leq 2(\eta pd + k)^2(4pd + 2)$ which is at most $k^{5(pd)^2}$.

Construction of G' : Let G' be the graph obtained by gluing the boundaried graphs $G[R[W_1, P_1]]$ and G_2 both having P_1^r as the boundary with the bijection corresponding to the gluing being an identity mapping from P_1^r to itself. See figure 3.6. We also similarly define G''' as the graph obtained by gluing the graphs $G[R[W_1, P_1]]$ and G'_2 both having P_1^r as the boundary with the bijection corresponding to the gluing being an identity mapping from P_1^r to itself.

We claim that for the graph G' , BRANCHING-SET with input $((G' \setminus K^{nr}, |K^r|, W_1, P_1^{nr}, U'), |X^r|, |K^r \setminus X^r|)$ returns a set \mathcal{R}' that intersects K^r where

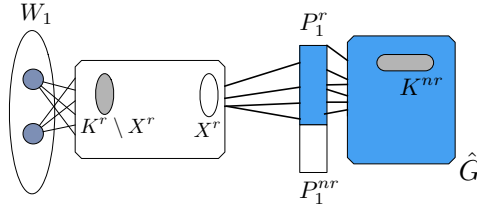


Figure 3.6: The graph $G' \setminus K^{nr}$ obtained from gluing the graphs $G[R[W_1, P_1]]$ and $\hat{G} \setminus K^{nr}$ along P_1^r where K^r is an optimal $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator

$U' = U \cup V(\hat{G}) \setminus P_1^r$. Suppose not. The set \mathcal{R}' by the definition of BRANCHING-SET procedure intersects any $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator $Z' \subseteq (V(G') \setminus U')$ of size $|K^r|$ in the graph $G' \setminus K^{nr}$ containing an $(|K^r \setminus X^r|, U')$ -important $W_1 - P_1^{nr}$ separator. Since K is a $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator in the graph $G[R[W_1, X]]$ with $K \cap U' = K^{nr}$, the set $K^r = K \setminus K^{nr}$ is a $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator of size $|K^r|$ for the graph $G[R[W_1, X]] \setminus K^{nr}$ with no vertices from U' . Let us turn the focus to the graph G''' where the degree 2 paths are not contracted. Since $G''' \setminus K^{nr}$ is an induced subgraph of the graph $G[R[W_1, X]] \setminus K^{nr}$, the set K^r is a $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator of size $|K^r|$ for the graph $G''' \setminus K^{nr}$ as well. In other words, every connected component of the graph $G''' \setminus K$ belongs to at least one of the graph class Π_i with $i \in [d]$.

Claim 3.2.1. K^r is a $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator of size $|K^r|$ for the graph $G' \setminus K^{nr}$.

The graph $G' \setminus K$ can be viewed as obtained from $G''' \setminus K$ by contracting some of the degree 2 paths of length more than $4pd + 2$ to $4pd + 2$. If we can prove that after doing this contraction in $G''' \setminus K$, the connected components still belongs to at least one of the graph class Π_i with $i \in [d]$, we are done.

Suppose not. Then there is a forbidden set C in one of the connected components of the graph $G' \setminus K$. Let us now uncontract the edges that we contracted. We will show that in the resulting graph $G''' \setminus K$, there exists a forbidden set C' which is isomorphic to $G[C]$. This contradicts our assumption that $G''' \setminus K$ is such that each of its components belongs to at least one of the graph classes Π_i for $i \in [d]$.

Let $C = \bigcup_{i \in [d]} C_i$ where C_i isomorphic to the graph $H_i \in \mathcal{F}_i$. Let α be one of the paths in the graph $G''' \setminus K$ with degree 2 vertices which was contracted to a path α' of length $4pd + 2$ in the graph $G' \setminus K$. The graph induced by $C_i \cap V(\alpha')$ is such that each connected component is a path of length at most p . In the path α too we can find a subset of vertices such that the graph induced by those vertices is isomorphic to the graph induced by $C_i \cap V(\alpha')$. Since α' has size $4pd + 2 > p$, no connected component of C_i for any $i \in [d]$ has both the endpoints of α' . Hence, if we replace $C_i \cap V(\alpha')$ with the corresponding subsets we identified in α , we get subsets C'_i so that the set $C' = \bigcup_{i \in [d]} C'_i$ in the graph $G''' \setminus K$ is a forbidden set isomorphic to C . The connectivity of C' is preserved as we only uncontract some edges. This contradicts that $G''' \setminus K$ is such that each of its components belongs to at least one of the graph classes Π_i for $i \in [d]$. This concludes the proof of the claim.

We now prove that X^r is a $(|K^r \setminus X^r|, U')$ -important separator in $G' \setminus K^{nr}$. This implies that the set returned from the recursive procedure \mathcal{R}' intersects K^r completing the proof of the lemma.

Claim 3.2.2. X^r is a $(|K^r \setminus X^r|, U')$ -important separator in $G' \setminus K^{nr}$.

We first prove that the set X^r is $(|K^r \setminus X^r|, U')$ -good $W_1 - P_1^{nr}$ separator in the graph $G''' \setminus K^{nr}$. We know that X is a $W_1 - P_1^{nr}$ separator in the graph G . Hence X^r is $W_1 - P_1^{nr}$ separator in the graph $G \setminus X^{nr}$. Since $G''' \setminus K^{nr}$ is an induced subgraph of $G \setminus X^{nr}$, we can conclude that X^r is a $W_1 - P_1^{nr}$ separator in the graph $G''' \setminus K^{nr}$. Since the graph $G' \setminus K^{nr}$ can be seen as obtained from $G''' \setminus K^{nr}$ by contracting some degree 2 paths with none of the edges with endpoints in X^r contracted, X^r is also a $W_1 - P_1^{nr}$ separator in the graph $G' \setminus K^{nr}$.

Suppose X^r is not a $(|K^r \setminus X^r|, U')$ -good $W_1 - P_1^{nr}$ separator in the graph $G' \setminus K^{nr}$. Then the graph $G' \setminus (K^{nr} \cup X^r)$ does not contain a $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator of size $|K^r \setminus X^r|$ with undeletable set U' . We claim that the set $K^r \setminus X^r$ is indeed such a $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -modulator. Suppose not. Then there exists a forbidden set C in the graph $G' \setminus (K^{nr} \cup X^r \cup (K^r \setminus X^r)) = G' \setminus K$. But this contradicts Claim 3.2.1.

Hence it remains to show that the set X^r is a $(|K^r \setminus X^r|, U')$ -important $W_1 - P_1^{nr}$ separator in the graph $G' \setminus K^{nr}$. Suppose this is not the case. Then there exists a $(|K^r \setminus X^r|, U')$ -good $W_1 - P_1^{nr}$ separator X' in the graph $G' \setminus K^{nr}$ that well-dominates X^r . We claim that if so, the set $\hat{X} = X' \cup X^{nr}$ is an (ℓ, U) -good $W_1 - W_2$ separator in the graph G with undeletable set U well-dominating X . This would contradict that X is an (ℓ, U) -important $W_1 - W_2$ separator in the graph G .

Let Y' be the set witnessing that X' is a $(|K^r \setminus X^r|, U')$ -good $W_1 - P_1^{nr}$ separator in the graph $G' \setminus K^{nr}$. Note that $X' \cup Y'$ is contained in the set $R[W_1, P_1]$ as it cannot contain vertices from the set $(V(\hat{G}) \setminus P_1^r) \subseteq U'$.

We now claim that $Y' \cup (K^{nr} \setminus X^{nr})$ is the set witnessing that \hat{X} is an (ℓ, U) -good $W_1 - W_2$ separator in the graph G . Suppose this is not the case. Then there exists a forbidden set C in the graph $G[R[W_1, \hat{X}]] \setminus (X' \cup X^{nr} \cup Y' \cup (K^{nr} \setminus X^{nr})) = G[R[W_1, \hat{X}]] \setminus K'$ where $K' = X' \cup Y' \cup K^{nr}$.

If $C \subseteq V(G') \setminus K'$, then the forbidden set C occurs in the graph $G' \setminus K'$ contradicting that X' is an $(|K^r \setminus X^r|, U')$ -good $W_1 - P_1^{nr}$ separator in the graph $G' \setminus K^{nr}$. Hence $C \cap ((R[W_1, \hat{X}] \setminus K') \setminus V(G'))$ is non-empty. Let $C = \bigcup_{i \in [d]} C_i$ where $G[C_i]$ is isomorphic to graphs $H_i \in \mathcal{F}_i$. Let $C_i^+ = C_i \cap NR[W_1, P_1]$ and $C^+ = \bigcup_{i \in [d]} C_i^+$. We have graphs $G[C_i^+]$ isomorphic to graphs $H_i[B_i]$ for subsets $B_i \subseteq V(H_i)$. Let $C_i^{P_1^r} = C_i \cap P_1^r$. Since $G[C]$ is connected, for vertices $a_i \in C_i$ and $a_j \in C_j$ with $i, j \in [d]$, there is a path P_{a_i, a_j} from a_i to a_j in the graph $G[R[W_1, \hat{X}]] \setminus K'$. Let the first and last vertices of P_1^r in P_{a_i, a_j} be the pair $t_{\rho(a_i, a_j)}$. Then for the tuple $\langle \mathcal{H}, \mathcal{B}_{\mathcal{H}}, \mathcal{P}_1^r, \mathcal{T}_{\mathcal{H}} \rangle$ where $\mathcal{H} = (H_1, \dots, H_d) \in \mathbb{H}$, $\mathcal{B}_{\mathcal{H}} = (B_1, \dots, B_d) \in \mathbb{B}_{\mathcal{H}}$, $\mathcal{P}_1^r = (C_1^{P_1^r}, \dots, C_d^{P_1^r}) \in \mathbb{P}_1^r$ and $\mathcal{T}_{\mathcal{H}} = (t_1, \dots, t_{\binom{[d]}{2}}) \in \mathbb{T}_{\mathcal{H}}$, there exists a forbidden set $C_M \subseteq (V_2 \cup V(G_1))$ of the graph $G \setminus K^{nr}$ which is marked.

Let $C_{M,i}$ be the set such that $G[C_{M,i}]$ is isomorphic to H_i . Also let $C_{M,i}^+ = C_{M,i} \cap (V(F))$. The set C_M can be viewed as replacing the vertices C^+ of C with C_M^+ .

We first claim that the set of vertices of C_M is present in the graph $G' \setminus K'$. Recall that

G' is obtained by contracting some degree 2 vertices of G''' which in turn is obtained by gluing the graphs $G[R[W_1, P_1]]$ and $G[V(F)]$. All the vertices of C_M is present in the graph $G''' \setminus K^{nr}$ as it contains all the marked vertices. In particular, all the vertices of C_M in the set V_2 are contained in the set M' , the set of vertices of all the marked forbidden sets contained in V_2 . When we transform G''' to G' , we only contract degree 2 paths in V_2 none of whose vertices belong to M' and hence C_M . Hence C_M is a present in the graph $G' \setminus K'$ as well.

If we can prove that C_M is in a connected component of $G' \setminus K'$, we can conclude that C_M is a forbidden set in the graph $G' \setminus K'$ contradicting that X' is a $(|K^r \setminus X^r|, U')$ -good $W_1 - P_1^r$ separator in the graph $G' \setminus K^{nr}$.

Suppose C_M is not in a connected component of $G' \setminus K'$. Then there exist a pair of vertices $u_1, u_2 \in C_M$ such that there is no path between u_1 and u_2 in the graph $G' \setminus K'$. But since C_M corresponds to the tuple $\langle \mathcal{H}, \mathcal{B}_{\mathcal{H}}, P_1^r, \mathcal{T}_{\mathcal{H}} \rangle$ with $\mathcal{T}_{\mathcal{H}} = (t_1 \dots t_{\binom{|C|}{2}}) \in \mathbb{T}_{\mathcal{H}}$, there exists a path P_{u_1, u_2} in the graph $G \setminus K^{nr}$ between u_1 and u_2 such that the first and last vertices of the path P_{u_1, u_2} intersecting P_1^r is the pair $t_{\rho(u_1, u_2)} = (\tau_1, \tau_2)$.

Let us also look at vertices $u'_1, u'_2 \in C$ such that in the isomorphism from C to C_M , u_i is mapped to u'_i for $i \in \{1, 2\}$. Since C is connected, there is a path $P_{u'_1, u'_2}$ between u'_1 and u'_2 in the graph $G[R[W_1, \hat{X}] \setminus K'$. Note that the forbidden sets C and C_M are both candidates for the marking procedure corresponding to the same tuple $\langle \mathcal{H}, \mathcal{B}_{\mathcal{H}}, P_1^r, \mathcal{T}_{\mathcal{H}} \rangle$. Hence we can assume that the path $P_{u'_1, u'_2}$ in the graph $G[R[W_1, \hat{X}] \setminus K'$ is such that the first and last vertices of the path $P_{u'_1, u'_2}$ intersecting P_1^r is the pair (τ_1, τ_2) .

We now identify all the vertices of P_1^r present in the path $P_{u'_1, u'_2}$ and partition them accordingly. Specifically, let us partition the path $P_{u'_1, u'_2}$ into a sequence of subpaths $\alpha_1, \dots, \alpha_q$ where the path α_1 is from u'_1 to τ_1 , the path α_q is from τ_2 to u'_2 , the path α_i where $1 < i < q$ has its endpoints in P_1^r and none of the internal vertices of the paths contain vertices of P_1^r . We aim to use the path P_{u_1, u_2} and the connectivities provided by the forest F in V_2 to construct a path between u_1 and u_2 in the graph $G' \setminus K'$ leading to a contradiction.

Let us now look at the cases based on whether $u_i, u'_i \in R[W_1, P_1]$ or not.

- Case 1, $u_1, u_2 \in R[W_1, P_1]$: Note that since both the vertices are in $V(G')$, we have $u_i = u'_i$ for $i \in \{1, 2\}$. The paths α_i which is not present in $G' \setminus K'$ are those whose internal vertices contains some vertices of $V_2 \setminus V(F)$. The paths α_1 and α_q are present in $G' \setminus K'$ as all its internal vertices including u_1 and u_2 are not in V_2 . Hence such paths α_i have both its endpoints in P_1^r . Also since P_1^r separates $R(W_1, P_1)$ from $V_2 \setminus P_1^r$, all paths α_i with $1 < i < q$ are such that all its internal vertices are either in $R(W_1, P_1)$ or in $V_2 \setminus P_1^r$. The former kind of paths are also present in $G' \setminus K'$.

Hence the path α_i that contains vertices of $V_2 \setminus F$ are such that its endpoints are in P_1^r and all its internal vertices in V_2 . The forest F preserved connectivities of vertices in P_1^r within V_2 including the endpoints of α_i . Hence we can replace α_i with the unique path between its endpoints in the forest F . Note that such a path is disjoint from K' and hence is present in $G' \setminus K'$.

By replacing all such paths α_i with those in F , we get a walk from u_1 to u_2 in the graph $G' \setminus K'$.

- Case 2, $u_1 \in R[W_1, P_1], u_2 \in V_2 \setminus P_1^r$: In this case, we have $u_1 = u'_1$. But it could be the case that $u_2 \neq u'_2$. Also it could be that u'_2 is a vertex not in $G' \setminus K'$.

Like we did in the previous case, we could replace all the paths α_i to paths in $G' \setminus K'$ using the forest F . Hence we have a path from u_1 to τ_2 in the graph $G' \setminus K'$. We now focus on the path α_q . We know that there is a subpath from τ_2 to u_2 in the path P_{u_1, u_2} in the graph $G \setminus K^{nr}$. Since τ_2 is the last vertex of P_1^r in the path, we know that this subpath is contained in the set V_2 . Since F is a forest that preserved connectivities of vertices between M' and P_1^r in the set V_2 , there is a path from τ_2 to u_2 in the forest F . Again note that such a path is disjoint from K' and hence is present in $G' \setminus K'$. We replace α_q with this path in F to get a walk from u_1 to u_2 in the graph $G' \setminus K'$.

- Case 3, $u_2 \in R[W_1, P_1], u_1 \in V_2 \setminus P_1^r$: This case is symmetric to the previous case and

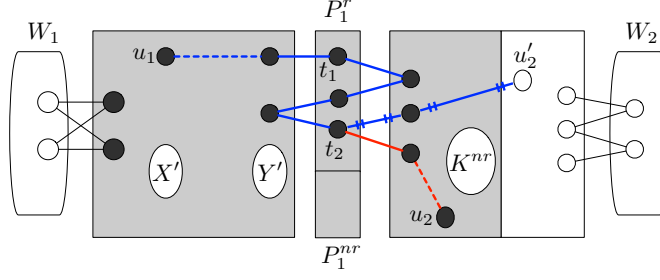


Figure 3.7: A demonstration of how $u_1, u_2 \in C_M$ are connected in the graph $G' \setminus K'$ (denoted by the grey region). We know from the marking procedure that both C and C_M are such that paths between vertices corresponding to u_1 and u_2 have its first and last vertex of P_1^r as t_1 and t_2 . We replace the path between t_2 and u_2' with the path between t_2 and u_2 guaranteed from the forest F .

the proof goes accordingly.

- Case 4, $u_1, u_2 \in V_2 \setminus P_1^r$: In this case, it could be that $u_2 \neq u_2'$ and $u_2 \neq u_2'$. Also it could be that the vertices u_1' and u_2' are not in $G' \setminus K'$.

We replace the path α_1 to one in the forest as we done for α_q in Case 2. Since τ_1 is the first vertex of P_1^r in the subpath between u_1 and τ_1 in the path P_{u_1, u_2} , such a path is contained in the set V_2 . The forest F preserves the connectivity between u_1 and τ_1 in V_2 . Hence we can replace the path α_1 with the one in F which is disjoint from K' .

The other paths α_i are replace similarly as in Case 2 to get a walk from u_1 to u_2 in the graph $G' \setminus K'$.

Hence the graph $G[C_M]$ is connected in the graph $G' \setminus K'$. Hence C_M is a forbidden set in the graph $G' \setminus K'$ contradicting that X' is $(|K^r \setminus X^r|, U')$ -good $W_1 - P_1^{nr}$ separator in the graph $G' \setminus K^{nr}$. Hence $|X^r|$ is $(|K^r \setminus X^r|, U')$ -important $W_1 - P_1^{nr}$ separator in the graph $G' \setminus K^{nr}$. This concludes the proof the claim and thereby the lemma. \square

From Lemma 3.2.10, we can infer using induction hypothesis that the BRANCHING-SET procedure called recursively with input as $((G' \setminus \Delta(\hat{G}), |K^r|, W_1, P_1^{nr}, U \cup V(\hat{G}) \setminus P_1^r), |X^r|, |K^r \setminus X^r|)$ returns a set \mathcal{R}' that intersects K^r . Since $K^r \subseteq Z$, we can conclude that \mathcal{R}' intersects Z as well. This completes the correctness of the BRANCHING-SET procedure.

Bounding the set \mathcal{R} : Let us look at the recursion tree of the BRANCHING SET procedure. The value of λ drops at every level of the recursion tree. Since $\lambda \leq k$, the depth of the tree is bounded by k . The number of branches at each node is at most $k^3 \cdot 2^k \cdot k! \cdot 2^{k^{10(pd)^2}}$ (k^3 for choice of λ', j and ℓ' , 2^k for choice of P_i' , $k!$ for the choice of the bijection δ and $2^{k^{10(pd)^2}} \geq 2^{\binom{k^{5(pd)^2}}{2}} \binom{k^{5(pd)^2}}{k+1}$ for the size of \mathcal{H}). Since, at each internal node, we add at most $2k$ vertices (corresponding to $P_1 \cup P_2$), we can conclude that the size of \mathcal{R} is bounded by $2^{k^{10(pd)^2+2}}$.

Let $\mathcal{R}_{\lambda', \ell'}$ denote the set returned by BRANCHING-SET procedure with input $(\mathcal{I}, \lambda', \ell')$. We define \mathcal{R} as the union of the sets $\mathcal{R}_{\lambda', \ell'}$ for all possible values of λ' and ℓ' . After this, in the MAIN-ALGORITHM procedure we simply branch on every vertex v of \mathcal{R} creating new instances $(G - v, k - 1, W_1, W_2)$ of DISJOINT FINITE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -VDC. If $k < 0$, we return NO. If Reduction Rule 2 applies, we use it to reduce the instance. If this results in a non-separating instance with $W = W_1 \cup W_2$, we apply the algorithm in Lemma 3.2.5 to solve the instance. Else we recursively run MAIN-ALGORITHM on the new instance.

Bounding running time of MAIN-ALGORITHM: We now bound the running time $T(k)$ for MAIN-ALGORITHM for the instance $\mathcal{I} = (G, k, W_1, W_2)$.

The depth of the branching tree is bounded by k and the branching factor at each node is $|\mathcal{R}| \leq 2^{k^{10(pd)^2+2}}$. Let $\gamma = 10(pd)^2$. We have the number of search nodes in the branching tree to be bounded by $(2^{k^{\gamma+2}})^k = 2^{k^{\gamma+3}}$. In each node, the time spent is $Q(k) + 2^{\eta k} n^\delta$ where $Q(k)$ is the running time for the BRANCHING-SET for the instance $(\mathcal{I}, \lambda, \ell)$ and $2^{\eta k} n^\delta$ is the time spent in the preprocessing stage for some constants η and δ . Hence we have $T(k) = 2^{k^{\gamma+3}} (Q(k) + 2^{\eta k} n^\delta)$.

Let us focus on the search tree for the BRANCHING-SET procedure. We know that the depth of the tree is bounded by k and the branching factor is bounded by $2^{k^{\gamma+2}}$. Again, the number of search nodes in the branching tree to be bounded by $(2^{k^{\gamma+2}})^k = 2^{k^{\gamma+3}}$. The time spent at each node is $\log n T(k-1) + 2^{k^\gamma} n^\epsilon$. The $\log n T(k-1)$ term comes from the at most $\log n$ many calls of sub-instances of MAIN-ALGORITHM called with strictly

smaller values of k . The $2^{k^\gamma} n^\varepsilon$ for some constant ε term comes from the other processing which includes the algorithm to enumerate graphs of size at most $2^{k^{5(pd)^2}}$. Hence we have $Q(k) = 2^{k^{\gamma+3}} (\log n T(k-1) + 2^{k^\gamma} n^\varepsilon)$.

We now substitute the value of $Q(k)$ above in the equation for $T(k)$ to get

$$\begin{aligned} T(k) &= 2^{k^{\gamma+3}} \left[2^{k^{\gamma+3}} (\log n T(k-1) + 2^{k^\gamma} n^\varepsilon) + 2^{\eta k} n^\delta \right] \\ &= 2^{2k^{\gamma+3}} \log n T(k-1) + 2^{2k^{\gamma+3} + k^\gamma} n^\varepsilon \end{aligned}$$

where assuming $\varepsilon > \delta$ (if not, set $\varepsilon = \delta + 1$), we can remove the term containing n^δ . We now have a recurrence on $T(k)$. We now prove by induction that $T(k) \leq 2^{3k^{\gamma+4}} (\log n)^k n^\varepsilon$. The base case is true as $T(0) \leq n^\varepsilon$. Assume the statement holds true for $1 \leq i \leq k-1$. Substituting the values for $T(k-1)$ in the recurrence for $T(k)$, we have

$$\begin{aligned} T(k) &\leq 2^{2k^{\gamma+3}} \log n \left[2^{3(k-1)^{\gamma+4}} (\log n)^{k-1} n^\varepsilon \right] + 2^{2k^{\gamma+3} + k^\gamma} n^\varepsilon \\ &\leq 2^{2k^{\gamma+3}} \log n \left[2^{3(k-1)k^{\gamma+3}} (\log n)^{k-1} n^\varepsilon \right] + 2^{3k^{\gamma+3}} n^\varepsilon \\ &= 2^{3k \cdot k^{\gamma+3} - 3k^{\gamma+3} + 2k^{\gamma+3}} (\log n)^k n^\varepsilon + 2^{3k^{\gamma+3}} n^\varepsilon \\ &\leq 2^{3k \cdot k^{\gamma+3} - 1} (\log n)^k n^\varepsilon + 2^{3k^{\gamma+3}} n^\varepsilon \\ &\leq 2^{3k \cdot k^{\gamma+3}} (\log n)^k n^\varepsilon \end{aligned}$$

By observing that $(\log n)^k \leq (k \log k)^k + n$, we get $T(k) = 2^{4k^{\gamma+4}} n^{\varepsilon+1}$.

Lemma 3.2.11. DISJOINT FINITE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -VDC can be solved in $2^{q(k)} n^{O(1)}$ time where $q(k) = 4k^{10(pd)^2+4} + 1$.

Proof. Let (G, k, W) be the instance of DISJOINT FINITE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ -VDC. We first apply Lemma 3.2.5 to see if there is a non-separating solution for the instance. If not, we branch over all 2^{k+1} subsets $W_1 \subset W$ and for each such choice of W_1 , apply MAIN-ALGORITHM procedure with input $(G, k, W_1, W_2, \emptyset)$ to check if $(G, k, W_1, W_2 = W \setminus W_1)$ has a solution containing a $W_1 - W_2$ separator. The correctness and running time follows

from those of Lemma 3.2.5 and correctness of the MAIN-ALGORITHM procedure. \square

As mentioned in Section 3.2.1, the time taken to solve FINITE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ VERTEX DELETION is $2^{k+1} \cdot 2^{q(k)} n^{\mathcal{O}(1)} = 2^{q(k)+1} n^{\mathcal{O}(1)}$. This proves Theorem 5 as $q(k)$ is a polynomial in k with p, d as constants.

3.3 Conclusion

We have initiated a study on vertex deletion problems to scattered graph classes and showed that the problem is non-uniformly FPT when there are a finite number of graph classes, the deletion problem corresponding to each of the finite classes is known to be FPT and the properties that a graph belongs to each of the classes is expressible in CMSO logic. Furthermore, we show that in the case where each graph class has a finite forbidden set, the problem is fixed-parameter tractable by a $2^{\text{poly}(k)} n^{\mathcal{O}(1)}$ time algorithm. The existence of a polynomial kernel for these cases are natural open problems. Exploring the fixed-parameter tractability of scattered versions of edge deletion and edge contraction problems is also open. In the traditional deletion setting, for most cases the techniques that helped to design algorithms for vertex deletion problems also works for edge deletion problems with minor changes. Hence, it would be interesting to see if the techniques in this chapter can be applied to scattered edge deletion problems with slight modifications.

In a recent work, Agrawal et al. [2] generalized Theorem 4 by showing ELIMINATION DISTANCE TO \mathcal{H} and TREEWIDTH DECOMPOSITION TO \mathcal{H} are FPT when \mathcal{H} is a scattered graph class satisfying the conditions for INDIVIDUALLY TRACTABLE $(\Pi_1, \Pi_2, \dots, \Pi_d)$ VERTEX DELETION.

Chapter 4

Faster Algorithms for Pairs of Scattered Graph Classes

In this chapter, we do a deep dive on pairs of specific graph classes (Π_1, Π_2) in which we would like the connected components of the resulting graph to belong to, and design simpler and more efficient FPT algorithms.

We look at specific variants of the following problem.

Π_1 OR Π_2 DELETION

Input: An undirected graph $G = (V, E)$, two hereditary graph classes Π_1 and Π_2 with \mathcal{F}_i as the forbidden family for graphs whose each connected component belongs to Π_i for $i \in \{1, 2\}$.

Parameter: k

Question: Is there a set $S \subseteq V(G)$ of size at most k such that every connected component of $G - S$ is in Π_1 or in Π_2 ?

Note that the forbidden families \mathcal{F}_i are for graphs whose each connected component is a graph belonging to Π_1 for $i \in \{1, 2\}$. It is not the forbidden family of graphs associated to the graph class Π_i . This distinction does not make a difference for most of the popular

graph classes as the union of connected components of such graph classes still belong to the graph class. Examples include bipartite graphs, chordal graphs, planar graphs, interval graphs and forests. But it is important for classes such as cliques and split graphs. The forbidden family for cliques is the singleton graph $2K_1$. But if the graph class is such that each connected component is a clique, $2K_1$ is present by taking a single vertex from two different components of the graph. The forbidden family in this case can be proven to be the singleton graph P_3 . In our definition of Π_1 OR Π_2 DELETION, when the graph class Π_1 is the class of cliques $\mathcal{F}_1 = \{P_3\}$ is the forbidden family of graphs where each component is clique.

We describe a general algorithm for Π_1 OR Π_2 DELETION under some conditions which covers pairs of several graph classes. While the specific conditions on the pairs of classes to be satisfied by this algorithm are somewhat technical and are explained in Section 4.3.2 and 4.3.3.2, we give a high-level description here.

We first make the reasonable assumption that the vertex deletion problems to the graph class Π_1 and to Π_2 have FPT algorithms. As we want every connected component of the graph after removing the solution vertices to be in Π_1 or in Π_2 , any pair of forbidden subgraphs $H_1 \in \mathcal{F}_1$ and $H_2 \in \mathcal{F}_2$ cannot both be in a connected component of G . Let us look at such a component C with $J_1, J_2 \subseteq V(C)$ such that $G[J_i]$ is isomorphic to H_i for $i \in \{1, 2\}$ and look at a path P between the sets J_1 and J_2 . Assuming that the graphs in families \mathcal{F}_1 and \mathcal{F}_2 are connected graphs, we can conclude that the solution has to hit the set $J_1 \cup J_2 \cup P$ allowing a branching on such sets.

But if the path is too large, such a branching does not lead to efficient algorithms. The generalization comes up from our observation that for certain pairs of graph classes, if we focus on a pair of forbidden subgraphs $H_1 \in \mathcal{F}_1$ and $H_2 \in \mathcal{F}_2$ that are “closest” to each other, then there is always a solution that does not intersect the shortest path P between them. This helps us to branch on the vertex sets of these forbidden graphs. However, note that the forbidden graphs may have unbounded sizes. We come up with a notion of *forbidden*

pair (Definition 4.3.2 in Section 4.3.1) and show that there are pairs of graph classes that have a finite number of forbidden pairs even if each of them has infinite forbidden sets. For some such pairs, we can bound the branching step to obtain the FPT algorithm.

4.1 Preliminaries

We introduce some notations and observations on Π_1 OR Π_2 DELETION that we use in this chapter. Throughout this chapter, we assume that the graphs in the forbidden families \mathcal{F}_1 and \mathcal{F}_2 associated to Π_1 OR Π_2 DELETION are connected which is true for most of the well-known graph classes. We use $\Pi_{(1,2)}$ to denote the class of graphs whose connected components are in the graph classes Π_1 or Π_2 .

Definition 4.1.1 (Minimal Forbidden Family). *A forbidden family \mathcal{F} for a graph class Π is said to be minimal if for all graphs $H \in \mathcal{F}$, we have that $\mathcal{F} - \{H\}$ is not a forbidden family for Π .*

Let $\mathcal{F}_1 \times \mathcal{F}_2 = \{(H_1, H_2) : H_1 \in \mathcal{F}_1 \text{ and } H_2 \in \mathcal{F}_2\}$. The following characterization for $\Pi_{(1,2)}$ is easy to see.

Observation 4.1.1. *A graph G is in the graph class $\Pi_{(1,2)}$ if and only if no connected component C of G contains H_1 and H_2 as induced graphs in C , where $(H_1, H_2) \in \mathcal{F}_1 \times \mathcal{F}_2$.*

Let $J_1, J_2 \subseteq V(G)$ such that $G[J_i]$ is isomorphic to graphs H_i for $i \in \{1, 2\}$ and $(H_1, H_2) \in \mathcal{F}_1 \times \mathcal{F}_2$. We call the sets J_1 and J_2 as the vertex sets of the pair (H_1, H_2) .

In the following lemma, we show that any solution to Π_1 OR Π_2 DELETION must either hit the vertex sets of a pair in $\mathcal{F}_1 \times \mathcal{F}_2$ or a path connecting them.

Lemma 4.1.1. *Let $J_1, J_2 \subseteq V(G)$ such that $G[J_i]$ is isomorphic to graphs H_i for $i \in \{1, 2\}$ and $(H_1, H_2) \in \mathcal{F}_1 \times \mathcal{F}_2$. Let P be a path between J_1 and J_2 in the graph G . Then any solution for Π_1 OR Π_2 DELETION with input graph G contains one of vertices in the set $J_1 \cup J_2 \cup P$.*

Proof. Suppose this is not the case. Since the graphs H_1 and H_2 are connected, the graph G' induced by the set $J_1 \cup J_2 \cup P$ is a connected subgraph of G . If a solution X does not intersect $J_1 \cup J_2 \cup P$, then G' occurs in a connected component C of the remaining graph $G - X$. The presence of graphs H_1 and H_2 in C implies that C is neither in Π_1 nor in Π_2 giving a contradiction that X is a solution. \square

We use the following reduction rule for Π_1 OR Π_2 DELETION whose correctness easily follows.

Reduction Rule 3. *If a connected component C of G is in Π_1 or in Π_2 , then delete C from G . The new instance is $(G - V(C), k)$.*

4.2 FINITE Π_1 OR Π_2 DELETION with forbidden paths

In this section, we restrict the problem to the case where both the forbidden families \mathcal{F}_1 and \mathcal{F}_2 are finite and there exists a path P_α in one of the families, say \mathcal{F}_1 where α is some constant. Observe that for several natural graph classes (like cluster graphs, edgeless graphs, split cluster graphs, cographs) paths above a certain length is forbidden.

We define the problem below.

FINITE Π_1 OR Π_2 DELETION WITH PATH

Input: An undirected graph G , and an integer k . Furthermore, for a fixed integer α , the path $P_\alpha \in \mathcal{F}_1$.

Question: Does G have a set S of at most k vertices such that every connected component of $G - S$ is in Π_1 or in Π_2 ?

Since both \mathcal{F}_1 and \mathcal{F}_2 are finite, the set $\mathcal{F}_1 \times \mathcal{F}_2$ is also finite.

Let us look at a pair (H_1, H_2) such that there exist copies of H_1 and H_2 in G as induced subgraphs with the distance between the vertex sets being the smallest among all the pairs

in $\mathcal{F}_1 \times \mathcal{F}_2$. We call such a pair the *closest pair*. We claim below that this distance is bounded by α .

Lemma 4.2.1. *Let $(H_1, H_2) \in \mathcal{F}_1 \times \mathcal{F}_2$ be a closest pair in the graph G and let (J_1, J_2) be a pair of vertex subsets corresponding to the pair. Let P be a shortest path between J_1 and J_2 . Then $|V(P)| \leq \alpha$.*

Proof. Suppose this is not the case. Then let us look at the set J'_1 of the last α vertices of P which is isomorphic to $P_\alpha \in \mathcal{F}_1$. Then the pair of vertex subsets (J'_1, J_2) corresponds to the pair (P_α, H_2) in the graph G with the distance between them as zero. This contradicts that (J_1, J_2) is the vertex subsets of the closest pair. \square

Hence, we have the following branching rule for closest pairs where we branch on the vertex subsets plus the vertices of the path. The correctness follows from Lemma 4.1.1.

Branching Rule 2. *Let (J^*, T^*) be the vertex subsets of a closest pair $(H_1, H_2) \in \mathcal{F}_1 \times \mathcal{F}_2$. Let P^* be a path corresponding to this forbidden pair. Then for each $v \in J^* \cup T^* \cup P^*$, we delete v and decrease k by 1, resulting in the instance $(G - v, k - 1)$.*

Using this branching rule, we have an FPT algorithm for FINITE Π_1 OR Π_2 DELETION WITH PATH. Let d_i be the size of a maximum sized finite forbidden graph in \mathcal{F}_i for $i \in \{1, 2\}$. Let $c = d_1 + d_2 + \alpha - 2$.

Theorem 8. FINITE Π_1 OR Π_2 DELETION WITH PATH *can be solved in $\mathcal{O}^*(c^k)$ time.*

Proof. We describe our algorithm as follows. Let (G, k) be an input instance of FINITE Π_1 OR Π_2 DELETION WITH PATH. We exhaustively apply Reduction Rule 3 and Branching Rule 2 in sequence to get an instance (G', k') . The algorithm finds the closest pair to apply Branching Rule 2 by going over all pairs in $(H_1, H_2) \in \mathcal{F}_1 \times \mathcal{F}_2$ and going over all subsets of size $|V(H_1)| + |V(H_2)|$ of the graph (which is still a polynomial in n) and checking the distance between them.

Every component of G' is such that it is \mathcal{F}_1 -free or \mathcal{F}_2 -free or in other words in Π_1 or Π_2 . Hence if $k > 0$, we return no-instance. Otherwise, we return yes-instance. Since the largest sized obstruction in these rules is at most $c = d_1 + d_2 + \alpha - 2$, the bounded search tree of the algorithm has c^k nodes bounding the running time to $c^k \text{poly}(n)$. This completes the proof. \square

We now describe a family of graphs such that instead of focusing that each component is free of pairs in $\mathcal{F}_1 \times \mathcal{F}_2$, we can check whether the graph is free of graphs in this family.

Let \mathcal{F}' be the minimal family of graphs such that for each member $H \in \mathcal{F}'$, there exist subsets $J_1, J_2 \subseteq V(H)$ such that $H[J_i]$ is isomorphic to $H_i \in \mathcal{F}_i$ for $i \in \{1, 2\}$ and $d_H(J_1, J_2) \leq \alpha - 1$.

From Lemma 4.1.1, it can be inferred that any graph without any members from \mathcal{F}' as induced subgraphs belong to the graph class $\Pi_{1,2}$. Hence, the set \mathcal{F}' is the forbidden family for the graph class $\Pi_{1,2}$. We now use this family to give a kernel and approximation algorithm for FINITE Π_1 OR Π_2 DELETION WITH PATH.

The size of any member $H \in \mathcal{F}'$ is bounded by $c = d_1 + d_2 + \alpha - 2$. Suppose not. Then we can identify a vertex $v \in V(H)$ which is not part of J_1, J_2 and a path P of length at most $\alpha - 1$ between them. But then the graph $H \setminus \{v\}$ is also in \mathcal{F}' contradicting that \mathcal{F}' is minimal. This also proves that the size of \mathcal{F}' is bounded by $2^{\binom{c+1}{2}}$ which is the bound on the number of graphs of at most c vertices.

Theorem 9. FINITE Π_1 OR Π_2 DELETION WITH PATH admits a c -approximation algorithm, and a $\mathcal{O}(k^c)$ sized kernel.

Proof. We know that \mathcal{F}' is a finite forbidden family for $\Pi_{1,2}$ with any forbidden graph present in \mathcal{F}' has size at most $c = d_1 + d_2 + \alpha - 2$. Since d_1, d_2 and α are all constants, FINITE Π_1 OR Π_2 DELETION WITH PATH is an instance of an implicit c -HITTING SET problem [45]. We can get a c -approximation algorithm by finding induced graphs in G

which are isomorphic to any forbidden set in \mathcal{F}' and adding all its vertices (which is at most c) to the solution as usually done in implicit c -HITTING SET problems. Using Sunflower Lemma [45, 54], FINITE Π_1 OR Π_2 DELETION WITH PATH admits a kernel of size $\mathcal{O}(k^c)$. This completes the proof. \square

4.3 Π_1 OR Π_2 DELETION with a constant number of forbidden pairs

4.3.1 Forbidden Characterization for Π_1 OR Π_2 DELETION

Unfortunately, the algorithm in Section 4.2 does not work when at least one of the sets \mathcal{F}_1 or \mathcal{F}_2 is infinite as the family $\mathcal{F}_1 \times \mathcal{F}_2$ is no longer finite. But we observed that for many problems, branching on most of the pairs in $\mathcal{F}_1 \times \mathcal{F}_2$ could be avoided.

We aim to identify such ‘redundant’ pairs in $\mathcal{F}_1 \times \mathcal{F}_2$. Instead of ensuring that such pairs are absent in a graph for an instance of Π_1 OR Π_2 DELETION, we identify some graphs which are forbidden in such a graph. Since ensuring the absence of forbidden graphs comes in the familiar HITTING SET framework, a characterization for $\Pi_{(1,2)}$ using such forbidden graphs and remaining irredundant pairs would be useful.

For example, let $\mathcal{F}_1 = \{C_3, C_4\}$ and $\mathcal{F}_2 = \{D_4, C_4\}$ where D_4 is the graph obtained after removing an edge from K_4 . We have $\mathcal{F}_1 \times \mathcal{F}_2 = \{(C_3, D_4), (C_3, C_4), (C_4, D_4), (C_4, C_4)\}$. Since $C_4 \in \mathcal{F}_1 \cap \mathcal{F}_2$, the graph C_4 is forbidden for the graph class $\Pi_{1,2}$. Hence the pairs (C_4, C_4) , (C_3, C_4) and (C_4, D_4) are redundant by identifying that C_4 is forbidden. Now note that C_3 is an induced subgraph of the graph D_4 . Hence the pair (C_3, D_4) can also be made redundant by identifying that D_4 is forbidden for $\Pi_{(1,2)}$.

We now formalize such forbidden graphs in the graph class $\Pi_{(1,2)}$ by defining the notion of super-pruned family. Recall that if a family of graphs is *minimal*, no element of it is an

induced subgraph of some other element of the family.

Definition 4.3.1 (Super-Pruned Family). *An element of a super-pruned family $\text{sp}(\mathcal{G}_1, \mathcal{G}_2)$ of two minimal families of graphs \mathcal{G}_1 and \mathcal{G}_2 is a graph that (i) belongs to one of the two families and (ii) has an element of the other family as induced subgraph.*

The family $\text{sp}(\mathcal{G}_1, \mathcal{G}_2)$ can be obtained from an enumeration of all pairs in $\mathcal{G}_1 \times \mathcal{G}_2$ and adding the supergraph if one of the graphs is an induced subgraph of the other. The family obtained is made minimal by removing the elements that are induced subgraphs of some other elements.

For example, let (Π_1, Π_2) be (Interval, Trees), with the forbidden families $\mathcal{F}_1 = \{\text{net, sun, long claw, whipping top, } \dagger\text{-AW, } \ddagger\text{-AW}\} \cup \{C_i : i \geq 4\}$ (See Figure 4.2) and \mathcal{F}_2 as the set of all cycles. Note that all graphs C_i with $i \geq 4$ are in $\text{sp}(\mathcal{F}_1, \mathcal{F}_2)$ as they occur in both \mathcal{F}_1 and \mathcal{F}_2 . The remaining pairs of $\mathcal{F}_1 \times \mathcal{F}_2$ contain triangles from \mathcal{F}_2 . If the graph from \mathcal{F}_1 is a net, sun, whipping top, \dagger -AW or \ddagger -AW, it contains triangle as an induced subgraph. Hence these graphs are also in the family $\text{sp}(\mathcal{F}_1, \mathcal{F}_2)$.

We now show that graphs in $\text{sp}(\mathcal{F}_1, \mathcal{F}_2)$ are forbidden in the graph class $\Pi_{(1,2)}$.

Lemma 4.3.1. *If a graph G is in the graph class $\Pi_{(1,2)}$, then no connected component of G contains a graph in $\text{sp}(\mathcal{F}_1, \mathcal{F}_2)$ as induced subgraphs.*

Proof. Suppose a graph $H \in \text{sp}(\mathcal{F}_1, \mathcal{F}_2)$ occur as induced subgraph of a connected component C of G . From the definition of Super-Pruned Family, we can associate a pair $(H_1, H_2) \in \mathcal{F}_1 \times \mathcal{F}_2$ to H such that either H is isomorphic to H_1 and H_2 is an induced subgraph of H_1 or vice-versa. Without loss of generality, let us assume the former. Since $H_i \in \mathcal{F}_i$, we know that C is not in the graph class Π_i for $i \in \{1, 2\}$. This contradicts that G is in the graph class $\Pi_{(1,2)}$. □

Hence any pair containing a graph from $\text{sp}(\mathcal{F}_1, \mathcal{F}_2)$ are redundant. But $\text{sp}(\mathcal{F}_1, \mathcal{F}_2)$ does not capture all the pairs in $\mathcal{F}_1 \times \mathcal{F}_2$. We now define the following family to capture the

remaining pairs.

Definition 4.3.2 (Forbidden Pair Family). A forbidden pair family \mathcal{F}_p , of \mathcal{F}_1 and \mathcal{F}_2 , consists of all pairs $(H_1, H_2) \in \mathcal{F}'_p$ such that both $H_1 \notin \text{sp}(\mathcal{F}_1, \mathcal{F}_2)$ and $H_2 \notin \text{sp}(\mathcal{F}_1, \mathcal{F}_2)$.

For example, if Π_1 is the class of interval graphs and Π_2 is the class of forests, we have already shown that $\text{sp}(\mathcal{F}_1, \mathcal{F}_2)$ contains all the graphs in \mathcal{F}_1 except long-claw. The only remaining pair is (long-claw, triangle) and the singleton set contain this pair forms the forbidden pair family.

Now we characterize $\Pi_{(1,2)}$ based on the super-pruned family and the forbidden pair family associated with \mathcal{F}_1 and \mathcal{F}_2 as follows. This is used in the algorithms in Section 4.3.

Lemma 4.3.2. *The following statements are equivalent.*

- *Each connected component of G is either in Π_1 or Π_2 .*
- *The graph G does not contain graphs in the super-pruned family $\text{sp}(\mathcal{F}_1, \mathcal{F}_2)$ as induced subgraphs. Furthermore, for pairs (H_1, H_2) in the forbidden pair family of \mathcal{F}_1 and \mathcal{F}_2 , H_1 and H_2 both cannot appear as induced subgraphs in a connected component of G .*

Proof. To prove the forward direction, note that from Lemma 4.3.1, the G does not contain graphs in the super-pruned family $\text{sp}(\mathcal{F}_1, \mathcal{F}_2)$ as induced subgraphs. Hence, suppose that there exists a pair $(H_1, H_2) \in \mathcal{F}_p$ in a connected component χ of G . But then χ cannot be in Π_1 due to the presence of H_1 and cannot be in Π_2 due to the presence of H_2 giving a contradiction. To prove the converse, suppose that G contains a component χ which is neither in Π_1 nor in Π_2 . Then there exist graphs $H_1 \in \mathcal{F}_1$ and $H_2 \in \mathcal{F}_2$ occurring as induced subgraphs of χ . If H_1 occurs as an induced subgraph of H_2 or vice-versa, then the supergraph occurs in $\text{sp}(\mathcal{F}_1, \mathcal{F}_2)$ giving a contradiction. Else we have $H_1 \in \mathcal{F}_1 \setminus \text{sp}(\mathcal{F}_1, \mathcal{F}_2)$ and $H_2 \in \mathcal{F}_2 \setminus \text{sp}(\mathcal{F}_1, \mathcal{F}_2)$. Hence $(H_1, H_2) \in \mathcal{F}_p$ giving a contradiction. \square

We now define useful notions of forbidden sets and closest forbidden pairs for the graph class $\Pi_{(1,2)}$.

Definition 4.3.3. *We call a minimal vertex subset $Q \subseteq V(G)$ as a forbidden set corresponding to the graph class $\Pi_{(1,2)}$ if $G[Q]$ is isomorphic to a graph in $\text{sp}(\mathcal{F}_1, \mathcal{F}_2)$ or $G[Q]$ is connected and contains both H_1 and H_2 as induced subgraphs for some forbidden pair (H_1, H_2) of $\Pi_{(1,2)}$.*

Definition 4.3.4. *We say that a forbidden pair (H_1, H_2) is a closest forbidden pair in a graph G if there exists subsets $J_1, J_2 \subseteq V(G)$ such that $G[J_1]$ is isomorphic to H_1 , $G[J_2]$ is isomorphic to H_2 and the distance between J_1 and J_2 in G is the smallest among all such pairs over all forbidden pairs of \mathcal{F}_1 and \mathcal{F}_2 . We call the pair of vertex subsets (J_1, J_2) as the vertex subsets corresponding to the closest forbidden pair. We call a shortest path P between J_1 and J_2 as the path corresponding to the closest forbidden pair.*

4.3.2 The case with forbidden paths

We now aim to give an algorithm for Π_1 OR Π_2 DELETION when the forbidden families \mathcal{F}_1 and \mathcal{F}_2 be infinite but the forbidden pair family \mathcal{F}_p is finite. We also assume that $P_\alpha \in \mathcal{F}_1$.

Let us list the conditions that Π_1 OR Π_2 DELETION is required to satisfy.

1. The vertex deletion problems for the graph classes Π_1 and Π_2 are FPT with algorithms to the respective classes being \mathcal{A}_1 and \mathcal{A}_2 .
2. \mathcal{F}_p , the forbidden pair family of \mathcal{F}_1 and \mathcal{F}_2 is of constant size.
3. The path $P_\alpha \in \mathcal{F}_1$.

P_α -FREE- (Π_1, Π_2) -DELETION

Input: An undirected graph G , graph classes Π_1, Π_2 with associated forbidden families \mathcal{F}_1 and \mathcal{F}_2 such that Conditions 1 - 3 are satisfied and an integer k .

Question: Does G have a set S of at most k vertices such that every connected component of $G - S$ is either in Π_1 or in Π_2 ?

Since the forbidden pair set is finite, we have the following Branching Rule for closest forbidden pairs which is similar to Branching Rule 2 where we branch on the vertex subsets plus the vertices of the path. The correctness follows from Lemma 4.1.1.

Branching Rule 3. *Let (J^*, T^*) be the vertex subsets of a closest pair $(H_1, H_2) \in \mathcal{F}_1 \times \mathcal{F}_2$. Let P^* be a path corresponding to this forbidden pair. Then for each $v \in J^* \cup T^* \cup P^*$, we delete v and decrease k by 1, resulting in the instance $(G - v, k - 1)$.*

From here on, assume that (G, k) be an instance at which Reduction Rule 3 and Branching Rule 3 are not applicable. Note that any component of G is now free of forbidden pairs.

Let \mathcal{F}_p^1 denote the family of graphs H_1 where $(H_1, H_2) \in \mathcal{F}_p$. Similarly define \mathcal{F}_p^2 as the family of graphs H_2 where $(H_1, H_2) \in \mathcal{F}_p$. By the definition of forbidden pairs, every pair (H_1, H_2) with $H_i \in \mathcal{F}_p^i$ is the forbidden pair set \mathcal{F}_p . Hence a graph that does not contain any forbidden pairs is \mathcal{F}_p^1 -free or \mathcal{F}_p^2 -free. With this observation, the following results are easy to see.

Lemma 4.3.3. *Let C be a connected component of G that is \mathcal{F}_p^i -free for $i \in \{1, 2\}$. If $G[C]$ has no Π_i vertex deletion set of size k , then (G, k) is a no-instance. Otherwise, let X be a minimum Π_i vertex deletion set of $G[C]$. Then (G, k) is a yes-instance if and only if $(G - V(C), k - |X|)$ is a yes-instance.*

Proof. Suppose that the premise of the statement holds and $k' = k - |X|$.

(\Leftarrow) The backward direction is trivial. If $G - V(C)$ has a feasible solution S' of size at most k' . Then, we can add the minimum sized Π_i vertex deletion set X of G and output

$S' \cup X$ has a feasible solution of size $k' + |X| = k$.

(\Rightarrow) We prove the forward direction now. Suppose that S^* be a feasible solution of size at most k to (G, k) and let $Y = S^* \cap V(C)$. We prove that $D = (S^* \setminus Y) \cup X$ is also a feasible solution to (G, k) and $|Y| \geq |X|$. If we manage to prove that Y is a Π_i -deletion vertex set of C then we are done. This is because since X is a minimum Π_i -deletion vertex set of C , $|X| \leq |Y|$. Since the graph $C - Y$ is in the graph class Π_i , the connected components $G - D$ are still in Π_1 or Π_2 after replacing Y with X . This implies that D is a feasible solution to (G, k) and $|D| \leq |S^*|$.

We now prove that Y is indeed a Π_i -deletion vertex set of C . Suppose not. Then there exist a forbidden set Q in $C - Y$. Note that C does not contain any forbidden pairs and is \mathcal{G}_1 -free. Hence from Lemma 4.3.2, Q is isomorphic to a graph in $\text{sp}(\mathcal{F}_1, \mathcal{F}_2) \setminus \mathcal{G}_1$. But in that case, from the definition of Super-Pruned Family, any graph $H \in \text{sp}(\mathcal{F}_1, \mathcal{F}_2)$ contains an induced subgraph which is isomorphic to some graph in \mathcal{F}_i . Hence the presence of Q contradicts that Y is a Π_i -deletion set. This completes the proof. \square

We are ready to prove our main theorem statement of this section. Let $f(k) = \max\{f_1(k), f_2(k)\}$ where $\mathcal{O}^*(f_i(k))$ is the running time for the algorithm \mathcal{A}_i . Also let c be the maximum among the size of graphs in \mathcal{G}_1 and the integer $\max_{(H_1, H_2) \in \mathcal{F}_p} (|H_1| + |H_2| + \alpha - 2)$.

Theorem 10. P_α -FREE- (Π_1, Π_2) -DELETION can be solved in $\mathcal{O}^*(\max\{f(k), c^k\})$ -time.

Proof. We describe our algorithm as follows. Let (G, k) be an input instance of P_α -FREE- (Π_1, Π_2) -DELETION. We exhaustively apply Reduction Rule 3 and Branching Rule 3 in sequence to get an instance (G', k') . The algorithm finds the closest pair to apply Branching Rule 3 by going over all pairs in $(H_1, H_2) \in \mathcal{F}_p$ and going over all subsets of size at most $|V(H_1)| + |V(H_2)|$ of the graph (which is still a polynomial in n) and checking the distance between them. Since the largest sized obstruction in these rules is at most c , the bounded

search tree of the algorithm so far has $c^{k-k'}$ nodes. Hence, every component of G' is such that it is \mathcal{F}_p^1 -free but has graphs in \mathcal{F}_p^2 as induced subgraphs, or vice-versa. In the first case, we invoke the $\mathcal{O}^*(f_1(|X|))$ -time algorithm for Π_1 VERTEX DELETION on $G[C]$ to compute a minimum Π_1 vertex deletion set X of $G[C]$. In the second case, we invoke the $\mathcal{O}^*(f_2(|X|))$ -time algorithm for Π_2 VERTEX DELETION to compute a minimum Π_2 vertex deletion set X of $G[C]$. The correctness follows from Lemma 4.3.3. This creates the total number of nodes in the search tree to $c^{k-k'} f(k')$, bounding the running time to $c^{k-k'} f(k') \text{poly}(n)$. This completes the proof. \square

We now give an approximation algorithm for P_α -FREE- (Π_1, Π_2) -DELETION when for $i \in \{1, 2\}$, Π_i VERTEX DELETION has an approximation algorithm with approximation factor c_i .

Theorem 11. P_α -FREE- (Π_1, Π_2) -DELETION has a d -approximation algorithm where $d = \max\{c, c_1, c_2\}$.

Proof. Let G be the input graph. Let S_{OPT} be the minimum sized set such that in the graph $G - S_{OPT}$, every connected component is either in Π_1 or Π_2 . Let $|S_{OPT}| = OPT$.

Let us define the family \mathcal{S}_1 as follows. Initially $\mathcal{S}_1 = \emptyset$. In polynomial time, we find the closest forbidden pair (J^*, T^*) in G with P^* being a shortest path between the pair, add $J^* \cup T^* \cup P^*$ to \mathcal{S}_1 and delete $J^* \cup T^* \cup P^*$ from G . We repeat this step until it is no longer applicable. Let S_1 be the set of vertices that is present in any pair of graphs in \mathcal{S}_1 . From Lemma 4.1.1, we can conclude that any feasible solution of G must contain a vertex from each member of the family \mathcal{S}_1 . Since the members of \mathcal{S}_1 are pairwise disjoint, we have that $|S_{OPT} \cap S_1| \geq |\mathcal{S}_1|$.

Let $G' = G - S_1$. We now construct a set S_2 as follows. Let C_1, \dots, C_q be the connected components of G' . If a connected component C_i has no graphs in \mathcal{F}_p^j as induced subgraph for $j \in \{1, 2\}$, we apply the c_j -approximation algorithm for Π_j VERTEX DELETION on $G'[C_i]$ to obtain a solution Z_i . The correctness comes from Lemma 4.3.3. We have

$S_2 = \bigcup_{i \in [q]} Z_i$. Since $(S_{OPT} - S_1) \cap C_i$ is an optimal solution for the connected component C_i of G' , we have that $|Z_i| \leq (\max\{c_1, c_2\})|(S_{OPT} - S_1) \cap C_i|$ for all $i \in [q]$.

We set $S = S_1 \cup S_2$. We have

$$\begin{aligned}
|S| &= |S_1| + |S_2| \\
&\leq \left(\max_{(H_1, H_2) \in \mathcal{F}_p} (|H_1| + |H_2| + \alpha - 2) \right) |S_1| + \sum_{i=1}^q |Z_i| \\
&\leq c|S_{OPT} \cap S_1| + \\
&\quad \sum_{i=1}^q (\max\{c_1, c_2\})|(S_{OPT} - S_1) \cap C_i| \\
&\leq (\max\{c, c_1, c_2\})|S_{OPT}|
\end{aligned}$$

Thus we have a d -approximation algorithm for P_α -FREE- (Π_1, Π_2) -DELETION. □

We now give some examples of P_α -FREE- (Π_1, Π_2) -DELETION.

4.3.2.1 Cliques or K_t -free graph subclass

We focus on the case of Π_1 OR Π_2 DELETION when Π_1 is the class of cluster graphs (where every connected component of the graph is a clique) and Π_2 is any graph class such that the complete graph K_t for some constant t is forbidden in this graph and the problem Π_2 VERTEX DELETION is known to be FPT. We show that this problem is an example of P_α -FREE- (Π_1, Π_2) -DELETION. We will see later that Π_2 can be many of the popular classes including planar graphs, cactus graphs, t -treewidth graph.

Let us formalize the problem.

CLIQUE-OR- K_t -FREE- Π_2 DELETION

Input: An undirected graph $G = (V, E)$, an integer k and Π_2 is the graph class which is K_t -free for some constant t and Π_2 VERTEX DELETION has an FPT algorithm \mathcal{A} with running time $\mathcal{O}^*(f_2(k'))$ for solution size k' .

Parameter: k

Question: Is there $S \subseteq V(G)$ of size at most k such that every connected component of $G - S$ is either a clique graph, or in Π_2 ?

We now show that Conditions 1 - 3 are satisfied by CLIQUE-OR- K_t -FREE- Π_2 DELETION. The problems CLIQUE VERTEX DELETION and Π_2 VERTEX DELETION has $\mathcal{O}^*(1.27^k)$ [36] and $\mathcal{O}^*(f_2(k))$ time FPT algorithms parameterized by the solution size k respectively. Hence Condition 1 is satisfied by CLIQUE-OR- K_t -FREE- Π_2 DELETION. Since $P_3 \in \mathcal{F}_1$, Condition 3 is satisfied by CLIQUE-OR- K_t -FREE- Π_2 DELETION. The only condition remaining to be proven is Condition 2 which we do below.

Lemma 4.3.4. *Condition 2 is satisfied by CLIQUE-OR- K_t -FREE- Π_2 DELETION.*

Proof. Let us first infer what the forbidden pair family corresponding to CLIQUE-OR- K_t -FREE- Π_2 DELETION is. We have the forbidden family for Π_1 as $\mathcal{F}_1 = \{P_3\}$. Note that we don't know what the forbidden family \mathcal{F}_2 for the graph class Π_2 is. We only know that the graph K_t is present in \mathcal{F}_2 . A crucial observation is that this is all needed to infer that the forbidden pair family for CLIQUE-OR- K_t -FREE- Π_2 DELETION.

We know that a graph is P_3 -free if and only if it is a collection of cliques. Hence, every graph $H \in \mathcal{F}_2$ that is not a collection of cliques, contains $P_3 \in \mathcal{F}_1$ as induced graphs. Hence all such graphs H belong to the Super-Pruned family $\text{sp}(\mathcal{F}_1, \mathcal{F}_2)$. We also know that every graph in \mathcal{F}_2 is connected. Hence, the only graph in the minimal family \mathcal{F}_2 that is P_3 -free is the graph K_t . Hence the forbidden pair family is the set $\{(P_3, K_t)\}$ which is of size one. \square

Since all the conditions of P_α -FREE- (Π_1, Π_2) -DELETION is satisfied by CLIQUE-OR- K_t -FREE- Π_2 DELETION, we have the following theorem.

Theorem 12. CLIQUE-OR- K_t -FREE- Π_2 DELETION has an FPT algorithm with running time $\mathcal{O}^*(\max\{(t+2)^k, f_2(k)\})$.

Proof. Observe that (P_3, K_t) is the only forbidden pair for CLIQUE-OR- K_t -FREE- Π_2 DELETION, the path between the closest forbidden pair is always empty (or in other words, the vertex subsets of the pair intersect). Hence the branching factor the Branching Rule is at most $|V(P_3)| + |V(K_t)| - 1 = t + 2$.

Hence from Theorem 10, we have a $\mathcal{O}^*(\max\{(t+2)^k, f_2(k)\})$ time algorithm for CLIQUE-OR- K_t -FREE- Π_2 DELETION. \square

We also give an approximation algorithm for CLIQUE-OR- K_t -FREE- Π_2 DELETION. But we appropriately change the definition of CLIQUE-OR- K_t -FREE- Π_2 DELETION where instead of the assumption that Π_2 VERTEX DELETION has an FPT algorithm with running time $\mathcal{O}^*(f_2(k'))$ for solution size k' , we assume that Π_2 VERTEX DELETION has a polynomial time approximation algorithm with approximation factor f_2 .

Observing that $d = \max\{c, c_1, c_2\} = \max\{t+2, f_2\}$ from Theorem 11, we have the following theorem.

Theorem 13. CLIQUE-OR- K_t -FREE- Π_2 DELETION has an approximation algorithm with approximation factor $\max\{t+2, f_2\}$.

We now give examples for the graph class Π_2 in CLIQUE-OR- K_t -FREE- Π_2 DELETION resulting in FPT and approximation algorithms for the corresponding problems.

- Let Π_2 be the class of trees. Since triangles are forbidden in trees, we have $t = 3$. The problem Π_2 VERTEX DELETION corresponds to FEEDBACK VERTEX SET which has a $\mathcal{O}^*(3.618^k)$ time FPT algorithm [106] and a 2-approximation algorithm [11]. We notice here that the closest (P_3, C_3) pair always intersect on at least two vertices. Hence the branching factor of Branching Rule 3 can be improved to $t + 1 = 4$ in this case.

- Let Π_2 be the class of cactus graphs. The graph K_4 is forbidden in cactus graphs as it has two triangles sharing an edge. Hence we have $t = 4$. The problem Π_2 VERTEX DELETION corresponds to CACTUS VERTEX DELETION which has a $\mathcal{O}^*(26^k)$ time FPT algorithm [21].
- Let Π_2 be the class of planar graphs. Since K_5 is not planar, we have $t = 5$. The problem Π_2 VERTEX DELETION corresponds to PLANAR VERTEX DELETION which has an $k^{O(k)} \text{poly}(n)$ time FPT algorithm [96] and a $\log^{O(1)} n$ -approximation algorithm with running time $n^{O(\log n / \log \log n)}$ [100].
- Let Π_2 be the class of η -treewidth graphs. Since $K_{\eta+2}$ has treewidth $\eta + 1$, it is forbidden in such graphs. The problem Π_2 VERTEX DELETION corresponds to η TREEWIDTH VERTEX DELETION which has a $2^{O(k)} n^{O(1)}$ time FPT algorithm and a $\mathcal{O}(1)$ -approximation algorithm [70].

We have the following corollary.

Corollary 4.3.1. Π_1 OR Π_2 DELETION *when Π_1 is the class of cliques and Π_2 is the class of*

- *trees has an $\mathcal{O}^*(4^k)$ time FPT algorithm and a 4-approximation algorithm.*
- *cactus graphs has an $\mathcal{O}^*(26^k)$ time FPT algorithm.*
- *planar graphs has an $k^{O(k)} \text{poly}(n)$ time FPT algorithm and a $\log^{O(1)} n$ -approximation algorithm with running time $n^{O(\log n / \log \log n)}$.*
- *η -treewidth graphs has a $\max\{(\eta + 4)^k, 2^{O(k)}\} n^{O(1)}$ time FPT algorithm and a $\mathcal{O}(1)$ -approximation algorithm.*

4.3.2.2 Split or Bipartite Graphs

SPLIT-OR-BIPARTITE DELETION

Input: An undirected graph $G = (V, E)$, an integer k .

Parameter: k

Question: Is there $S \subseteq V(G)$ of size at most k such that every connected component of $G - S$ is either a split graph or bipartite?

The family \mathcal{F}_1 for graphs whose each connected component is a split graph is $\mathcal{F}_1 = \{C_4, C_5, P_5, \text{necktie}, \text{bowtie}\}$. [30]. A necktie is the graph with vertices $\{a, b, c, d, e\}$ where $\{a, b, e\}$ forms a triangle and $\{a, b, c, d\}$ forms a P_4 . A bowtie is the graph obtained from a necktie by adding the edge (b, d) . The family \mathcal{F}_2 for graphs whose each connected component is a bipartite graph is the set of odd cycles.

The problems SPLIT VERTEX DELETION and ODD CYCLE TRANSVERSAL has $\mathcal{O}^*(1.27^k)$ [49] and $\mathcal{O}^*(2.314^k)$ time [117] FPT algorithms parameterized by the solution size k respectively. Hence Condition 1 is satisfied by SPLIT-OR-BIPARTITE DELETION. Since $P_5 \in \mathcal{F}_1$, Condition 3 is satisfied by SPLIT-OR-BIPARTITE DELETION.

The graph C_5 is common in both families whereas P_5 is an induced subgraph of odd cycles C_i with $i \geq 7$. Hence the family $\text{sp}(\mathcal{F}_1 \times \mathcal{F}_2)$ contains of all members in \mathcal{F}_2 except C_3 . Since both necktie and bowtie contains triangles, they are part of $\text{sp}(\mathcal{F}_1 \times \mathcal{F}_2)$ as well. The only remaining pairs are (C_4, C_3) and (P_5, C_3) which forms the forbidden pair family \mathcal{F}_p . Since it is of size two, Condition 2 is satisfied.

Hence, we have the following corollary from the algorithms for P_α -FREE- (Π_1, Π_2) -DELETION. Observing that the largest obstruction set that we branch on is for the pair (C_4, C_3) with a path of length at most four between them, we have $c = 11$. The problem ODD CYCLE TRANSVERSAL has a $\log(OPT)$ approximation algorithm where OPT is the size of the optimal solution. This dominates the approximation factor of the algorithm

obtained from Theorem 11.

Corollary 4.3.2. SPLIT-OR-BIPARTITE DELETION *can be solved in $\mathcal{O}^*(11^k)$ -time and has a $\log(OPT)$ approximation algorithm where OPT is the size of the optimal solution.*

4.3.3 Algorithms for Π_1 OR Π_2 DELETION without forbidden paths

We have seen examples of Π_1 OR Π_2 DELETION where even though the families \mathcal{F}_i are infinite, we manage to come up with fast FPT algorithms. This is mainly thanks to Branching Rule 3 whose branching factor is bounded due to the fact that the path between them is bounded. We now look at examples of Π_1 OR Π_2 DELETION where paths are not present in the sets \mathcal{F}_i . Hence the path between the closest forbidden pair is no longer bounded. We observe that for certain pairs of graph classes, there is always an optimal solution that does not intersect the path.

We give a general algorithm for pairs of graph classes where we enforce this condition. We first look at the simple case of CLAW-FREE-OR-TRIANGLE-FREE DELETION as a precursor to the algorithm.

4.3.3.1 Claw-free or Triangle-Free graphs

We define the problem.

CLAW-FREE-OR-TRIANGLE-FREE DELETION

Input: An undirected graph $G = (V, E)$ and an integer k .

Parameter: k

Question: Is there $S \subseteq V(G)$ of size at most k such that every connected component of $G - S$ is either a claw-free graph, or a triangle-free graph?

The forbidden pair family corresponding to the graph class is of size one which is

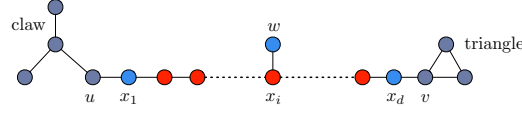


Figure 4.1: An illustration of a shortest path between a closest (claw, triangle) pair.

$\{(K_{1,3}, C_3)\}$. We now describe a branching rule corresponding to the closest forbidden pair in the graph.

Branching Rule 4. *Let (J^*, T^*) be the vertex subsets of a closest claw-triangle pair in a connected component of G , where $G[J^*]$ is isomorphic to a claw, and $G[T^*]$ is isomorphic to a triangle. Then for each $v \in J^* \cup T^*$, delete v and decrease k by 1, resulting in the instance $(G - v, k - 1)$.*

We now prove that Branching Rule 4 is safe. Let $P^* := x_0, x_1, \dots, x_{d-1}, x_d$ be a shortest path between J^* and T^* of length $d_G(J^*, T^*) = d$ with $x_0 = u \in J^*$ and $x_d = v \in T^*$. Let C be the connected component of the graph $G - (J^* \cup T^*)$ containing the internal vertices of P^* . We have the following lemma.

Lemma 4.3.5. *The graph corresponding to C is the path P^* without its end vertices. Furthermore, the only vertices of C adjacent to $J^* \cup T^*$ are x_1 and x_{d-1} which are only adjacent to vertices u and v respectively.*

Proof. Suppose that there is a vertex $w \in V(C) \setminus V(P^*)$ that is adjacent to a vertex $x_i \in V(P^*)$ with $1 \leq i \leq d - 1$. Let us look at the graph induced by the set of vertices $\{w, x_{i-1}, x_i, x_{i+1}\}$ in G . If w is adjacent to x_{i-1} , then the graph induced by the vertices $T' = \{w, x_{i-1}, x_i\}$ forms a triangle. Then since $d_G(J^*, T') < d_G(J^*, T^*)$, the pair (J^*, T') contradicts that (J^*, T^*) is the closest forbidden pair in the graph G . We can similarly prove that w is not adjacent to x_{i+1} . When w is not adjacent to both the vertices x_{i-1} and x_{i+1} , the graph induced by the vertices $J' = \{w, x_{i-1}, x_i, x_{i+1}\}$ forms a claw. Then since $d_G(J', T^*) < d_G(J^*, T^*)$, the pair (J', T^*) contradicts that (J^*, T^*) is the closest forbidden pair in the graph G .

Hence we conclude that there is no vertex $w \in V(C) \setminus V(P^*)$ that is adjacent to a vertex $x_i \in V(P^*)$ with $1 \leq i \leq d-1$. Since C is defined as the connected component of graph $G - (J^* \cup T^*)$ containing the internal vertices of P^* , the first statement of the claim follows.

The second statement is contradicted only when there is an edge from a vertex $x_i \in V(P^*)$ with $1 \leq i \leq d-1$ to the set $J^* \cup T^*$ apart from the edges (u, x_1) and (x_{d-1}, v) . If $2 \leq i \leq d-2$, this creates a shorter path P' between J^* and T^* via this edge giving a contradiction. For $i = 1$, in the case x_1 is adjacent to some vertex $w \in J^* \setminus \{u\}$, we can show that either the graph induced by the set of vertices $\{w, u, x_1, x_2\}$ is a claw or the graph induced by the set of vertices $\{w, u, x_1\}$ is a triangle, both contradicting that (J^*, T^*) is the closest forbidden pair in the graph G . For $i = d-1$, in the case x_{d-1} is adjacent to $w \in T^* \setminus \{v\}$, we can show that the graph induced by the vertices $\{w, x_{d-1}, v\}$ is a triangle contradicting that (J^*, T^*) is the closest forbidden pair in the graph G . This covers all the cases. \square

Lemma 4.3.6. *Branching Rule 4 is sound.*

Proof. Suppose not. In this case, all the optimal solutions for a CLAW-FREE-OR-TRIANGLE-FREE DELETION instance is such that it does not intersect $J^* \cup T^*$. Let $P^* := u, x_1, \dots, x_{d-1}, v$ be a shortest path between J^* and T^* of length $d_G(J^*, T^*) = d$. Since the graphs $G[J^*]$ and $G[T^*]$ cannot be in the same connected component after deleting the solution, all the optimal solutions X has to intersect the set of internal vertices of P^* .

We now claim that $X' = (X \setminus (P^* \setminus \{u\})) \cup \{v\}$ is also an optimal solution for G . Suppose not. Then there is a forbidden set Q such that $Q \cap X' = \emptyset$. Note that Q is a connected set and intersects some vertex $x_i \in P^* \setminus \{u, v\}$ with $1 \leq i \leq d-1$ as X is a feasible solution. The graph $G[Q]$ contain a forbidden pair for $(K_{1,3}, C_3)$ and hence contains a cycle. Hence Q is not fully contained in the connected component C of the graph $G \setminus (J^* \cup T^*)$ which is $P^* \setminus \{u, v\}$ from Lemma 4.3.5. Hence we conclude that Q contains some vertex outside $P^* \setminus \{u, v\}$ as well.

From Lemma 4.3.5, the only neighbors of C is u and v via x_1 and x_{d-1} respectively. Since $v \in X'$ and Q is connected, we can conclude that Q contains the subpath P' from u to x_i . We now claim that even after deleting the vertices of this subpath from Q except u , the set remains forbidden. This contradicts that Q is a forbidden set as it is not a minimal set.

Since Q is a forbidden set, it contains vertex subsets that are isomorphic to a claw and a triangle. From Lemma 4.4.3, we can conclude that none of the vertices of the subpath P' can be part of any triangle in G . None of these vertices can be part of a claw in G either as it contradicts that (J^*, T^*) is the closest forbidden pair. Since $v \in X'$ disconnects the path P^* , the subpath P' is not part of a path connecting a claw and a triangle either. Hence the set after removing the vertices of P' from Q is still a forbidden set contradicting that Q is minimal. \square

We are ready to give the algorithm for CLAW-FREE-OR-TRIANGLE-FREE DELETION.

Theorem 14. CLAW-FREE-OR-TRIANGLE-FREE DELETION *can be solved in $\mathcal{O}^*(7^k)$ time.*

Proof. Let (G, k) be an input instance of CLAW-FREE-OR-TRIANGLE-FREE DELETION. We exhaustively apply Reduction Rule 3 and Branching Rule 4 in sequence to get an instance (G', k') such that any component of G' is either claw-free or triangle-free. Note that finding the closest claw-triangle pair can be done by going over all subsets of size at most 7, checking if they do induce a claw and a triangle and finding the shortest path between them. The correctness follows from Lemma 4.3.6. If $k' < 0$, we return yes-instance. Else, we return yes-instance.

Let us look at the graph where the above rules are not applicable and $k' \geq 0$. We claim that the graph is such that every connected component is either claw-free or triangle-free. Suppose not. Then there exists a component C which contains both a claw and a triangle as induced subgraphs. This contradicts that Branching Rule 4 is no longer applicable from the presence of the closest such claw-triangle pair in C . This proves the correctness of the

algorithm.

Since we branch on a set of size at most 7 in Branching Rule 4, the bounded search tree of the algorithm has at most 7^k nodes. This bounds the running time to $7^k n^{\mathcal{O}(1)}$. This completes the proof. \square

We also give an approximation algorithm for CLAW-FREE-OR-TRIANGLE-FREE DELETION using similar ideas.

Theorem 15. CLAW-FREE-OR-TRIANGLE-FREE DELETION has a 7-approximation algorithm.

Proof. Let G be the input graph. The approximation algorithm for CLAW-FREE-OR-TRIANGLE-FREE DELETION is as follows. Let \mathcal{S}_1 be a family of sets initialized to \emptyset . We find a closest forbidden pair (J^*, T^*) in G' , add $J^* \cup T^*$ to \mathcal{S}_1 and delete $J^* \cup T^*$ from G' . We repeat this step until it is no longer applicable. Let S_{OPT} be the minimum sized set such that in the graph $G - S_{OPT}$, every connected component is either a claw-free graph or a triangle-free graph. Let $|S_{OPT}| = OPT$. Let S_1 be the set of vertices that is present in any set in \mathcal{S}_1 . From the safeness proof of Branching Rule 4, we can conclude that any feasible solution of G must contain a vertex from each set of the family \mathcal{S}_1 . Since the union of all the sets in \mathcal{S}_1 is a feasible solution, we have that $|S_1| = 7|\mathcal{S}_1| \leq 7|S_{OPT}|$.

Thus we have a 7-approximation algorithm for CLAW-FREE-OR-TRIANGLE-FREE DELETION. \square

4.3.3.2 Algorithm for SPECIAL INFINITE- (Π_1, Π_2) -DELETION

Now, we show that the algorithm ideas from CLAW-FREE-OR-TRIANGLE-FREE DELETION are applicable for a larger number of pairs of graph classes. Later in Section 4.4, we give other examples for pairs of graph classes where the same ideas work.

We define a variant of Π_1 OR Π_2 DELETION called SPECIAL INFINITE- (Π_1, Π_2) -DELETION satisfying the following properties.

1. The vertex deletion problems for the graph classes Π_1 and Π_2 are FPT with algorithms to the respective classes being \mathcal{A}_1 and \mathcal{A}_2 .
2. \mathcal{F}_p , the forbidden pair family of \mathcal{F}_1 and \mathcal{F}_2 is of constant size.
3. Let $(H_1, H_2) \in \mathcal{F}_p$ be a closest forbidden pair in the graph G with (J_1, J_2) being the vertex subsets corresponding to the pair. Let P be a shortest path between J_1 and J_2 . There is a family \mathcal{G}_1 such that

- \mathcal{G}_1 is a finite family of graphs of bounded-size (independent of the size of G), and
- in the graph G that is \mathcal{G}_1 -free, if a forbidden set Q intersects the internal vertices of P , then Q contains the right endpoint of P .

SPECIAL INFINITE- (Π_1, Π_2) -DELETION

Input: An undirected graph $G = (V, E)$, graph classes Π_1, Π_2 with associated forbidden families \mathcal{F}_1 and \mathcal{F}_2 such that Conditions 1 - 3 are satisfied and an integer k .

Parameter: k

Question: Is there a vertex set S of size at most k such that every connected component of $G - S$ is either in Π_1 or in Π_2 ?

Note that the first two conditions for the problem are the same as those in P_α -FREE- (Π_1, Π_2) -DELETION. Only the Condition 3 is changed which is tailored to prove the soundness of the branching rule we introduce.

Towards an FPT algorithm for SPECIAL INFINITE- (Π_1, Π_2) -DELETION, We give the following branching rule whose soundness is easy to see.

Branching Rule 5. *Let (G, k) be the input instance and let $Q \subseteq V(G)$ such that $G[Q]$ is isomorphic to a graph in \mathcal{G}_1 . Then, for each $v \in V(Q)$, delete v from G and decrease k by 1. The resulting instance is $(G - v, k - 1)$.*

From here on we assume that Branching Rule 5 is not applicable for G and so G is \mathcal{G}_1 -free. We now focus on connected components of G which contain forbidden pairs. We have the following branching rule.

Branching Rule 6. *Let (J^*, T^*) be the vertex subsets of a closest forbidden pair $(H_1, H_2) \in \mathcal{F}_p$. Then for each $v \in J^* \cup T^*$, we delete v and decrease k by 1, resulting in the instance $(G - v, k - 1)$.*

We now prove the correctness of the above branching rule.

Lemma 4.3.7. *Branching Rule 6 is safe.*

Proof. Let P^* be a shortest path between J^* and T^* with endpoints $u \in J^*$ and $v \in T^*$. Since $G[J^*]$ and $G[T^*]$ cannot occur in the same connected component after deleting the solution, the solution must intersect $J^* \cup T^* \cup P^*$. We now prove that there exists an optimal solution of (G, k) that does not intersect the internal vertices of P^* .

Let X be an optimal solution such that $X \cap (J^* \cup T^*) = \emptyset$. Since $X \cap (J^* \cup T^* \cup P^*) \neq \emptyset$, X must intersect the internal vertices of P^* . We claim that $X' = (X \setminus (P^* \setminus \{u\})) \cup \{v\}$ is also an optimal solution for G . Suppose not. Then there exists a forbidden set Q such that $X' \cap Q = \emptyset$. Since $X \cap Q \neq \emptyset$, we know that Q intersects the internal vertices of P^* . But then by Condition 3, we know that Q contains v as well giving a contradiction. \square

We now give the FPT algorithm SPECIAL INFINITE- (Π_1, Π_2) -DELETION which is the same algorithm in Theorem 10, but the Branching Rule 3 is replaced by Branching Rule 5 and Branching Rule 6 in sequence. The correctness comes from Lemmas 4.3.7 and 4.3.3.

Again we define $f(k) = \max\{f_1(k), f_2(k)\}$ where $\mathcal{O}^*(f_i(k))$ is the running time for the algorithm \mathcal{A}_i . Also let c be the maximum among the size of graphs in \mathcal{G}_1 and the integer

$\max_{(H_1, H_2) \in \mathcal{F}_p} (|H_1| + |H_2|)$. Note that the branching factor of Branching Rule 6 is reduced to $\max_{(H_1, H_2) \in \mathcal{F}_p} (|H_1| + |H_2|)$ as we do not branch on the vertices of the path between the vertex sets of the closest forbidden pair.

Theorem 16. SPECIAL INFINITE- (Π_1, Π_2) -DELETION *can be solved in*
 $\mathcal{O}^*(\max\{f(k), c^k\})$ -time.

We now give an approximation algorithm for SPECIAL INFINITE- (Π_1, Π_2) -DELETION when for $i \in \{1, 2\}$, Π_i VERTEX DELETION has an approximation algorithm with approximation factor c_i . The algorithm is similar to that of Theorem 11 with an additional primary step of greedily adding vertex subsets of induced graphs isomorphic to members in the family \mathcal{G}_1 to the solution. Note that any optimal solution should contain at least one of the vertices of each such vertex subset.

Theorem 17. SPECIAL INFINITE- (Π_1, Π_2) -DELETION *has a d -approximation algorithm where $d = \max\{c, c_1, c_2\}$.*

4.4 Examples of SPECIAL INFINITE- (Π_1, Π_2) -DELETION

Verifying whether Conditions 2 and 3 are satisfied for a general Π_1 and Π_2 is non-trivial. Hence we look at specific pairs of graph classes Π_1 and Π_2 can prove that they are examples of SPECIAL INFINITE- (Π_1, Π_2) -DELETION.

We start by showing that the problems CLAW-FREE-OR-TRIANGLE-FREE DELETION is indeed examples of SPECIAL INFINITE- (Π_1, Π_2) -DELETION.

Lemma 4.4.1. CLAW-FREE-OR-TRIANGLE-FREE DELETION *is an example of* SPECIAL INFINITE- (Π_1, Π_2) -DELETION.

Proof. We show that Conditions 1 - 3 are satisfied by CLAW-FREE-OR-TRIANGLE-FREE DELETION. The problems CLAW-TREE VERTEX DELETION and TRIANGLE VERTEX

DELETION has simple $\mathcal{O}^*(4^k)$ and $\mathcal{O}^*(3^k)$ time FPT algorithms parameterized by the solution size k via a simple branching on claws and triangles respectively. Hence Condition 1 is satisfied by CLAW-FREE-OR-TRIANGLE-FREE DELETION. The forbidden pair family for CLAW-FREE-OR-TRIANGLE-FREE DELETION is of size one which is $\{K_{1,3}, C_3\}$. Hence Condition 2 is satisfied. Finally, from Lemma 4.3.6, we can conclude that Condition 3 is satisfied as well.

Hence CLAW-FREE-OR-TRIANGLE-FREE DELETION is an example of SPECIAL INFINITE- (Π_1, Π_2) -DELETION. We have Let $f(k) = \max\{4^k, 3^k\} = 4^k$ and $c = \max_{(H_1, H_2) \in \mathcal{F}_p} (|H_1| + |H_2|) = 7$. Hence from Theorem 16, we have a $\mathcal{O}^*(7^k)$ time algorithm for CLAW-FREE-OR-TRIANGLE-FREE DELETION. This is the same running time obtained independently in Theorem 14.

We also have $d = \max\{c, c_1, c_2\} = 7$ giving a 7-approximation for CLAW-FREE-OR-TRIANGLE-FREE DELETION from Theorem 17, which is the same approximation factor obtained independently in Theorem 15. \square

We now give examples of other pairs of graph classes Π_1 and Π_2 whose scattered deletion problem is an example of SPECIAL INFINITE- (Π_1, Π_2) -DELETION. The core part in each of the cases below is establishing that Condition 3 is satisfied. We do so by establishing structural properties for the shortest path corresponding to the closest forbidden pair. Such properties vary for each case.

4.4.1 Interval or Trees

We define the problem.

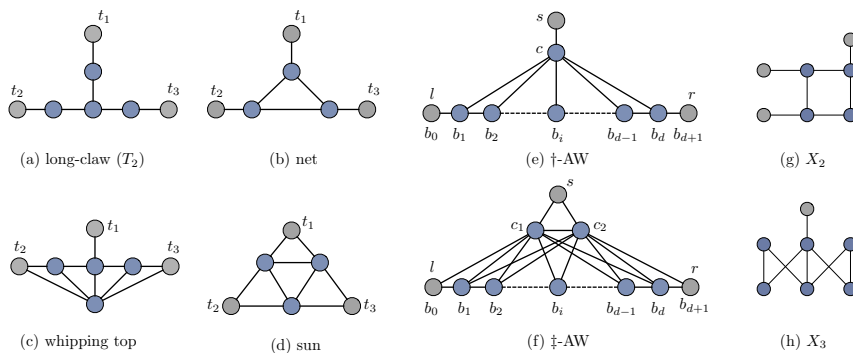


Figure 4.2: Obstructions for Graph Classes

INTERVAL-OR-TREE DELETION

Input: An undirected graph $G = (V, E)$ and an integer k .

Parameter: k

Question: Is there $S \subseteq V(G)$ of size at most k such that every connected component of $G - S$ is either an interval graph, or a tree?

We have the following forbidden subgraph characterization of interval graphs.

Lemma 4.4.2. ([109]) *A graph is an interval graph if and only if it does not contain net, sun, hole, whipping top, long-claw, †-AW, or ‡-AW as its induced subgraphs.*

See Figure 4.2 for an illustration of the graphs mentioned as forbidden subgraphs for interval graphs.

We now give a characterization for graphs whose every connected component is an interval graph or a tree. Recall from Section 4.3.1 that the forbidden pair family \mathcal{F}_p of this pair of graph classes is (long claw, triangle) and $\text{sp}(\mathcal{F}_1, \mathcal{F}_2)$ is {net, sun, hole, whipping top, †-AW, ‡-AW}. The following is a corollary from Lemma 4.3.2.

Corollary 4.4.1. *The following statements are equivalent.*

1. *Let G be a graph such that every connected component is either an interval graph or a tree.*
2. *G does not have any net, sun, hole, whipping top, †-AW, ‡-AW as its induced*

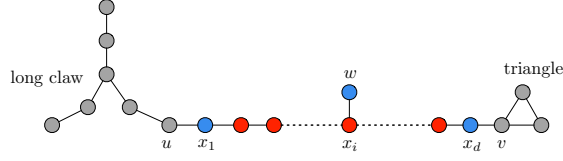


Figure 4.3: An illustration of a shortest path between a closest (long claw, triangle) pair.

subgraphs. Moreover, G cannot have long-claw and triangle as induced subgraphs in the same connected component.

We show that Conditions 1 - 3 are satisfied by INTERVAL-OR-TREE DELETION. The problems INTERVAL VERTEX DELETION and FEEDBACK VERTEX SET has $\mathcal{O}^*(10^k)$ [34] and $\mathcal{O}^*(3.618^k)$ [106] time FPT algorithms parameterized by the solution size k respectively. Hence Condition 1 is satisfied by INTERVAL-OR-TREE DELETION. The forbidden pair family for INTERVAL-OR-TREE DELETION is of size one which is the pair (long-claw, triangle). Hence Condition 2 is satisfied.

It remains to show that Condition 3 is satisfied for INTERVAL-OR-TREE DELETION. We define \mathcal{G}_1 be the family of graphs in $\text{sp}(\mathcal{F}_1, \mathcal{F}_2)$ of size at most 10.

Let (J^*, T^*) be the vertex subsets of a closest long-claw, triangle pair in a connected component of G , where J^* is a long-claw and T^* is a triangle. Let $P^* := x_0, x_1, \dots, x_{d-1}, x_d$ be a shortest path between J^* and T^* of length $d_G(J^*, T^*) = d$ with $x_0 = u \in J^*$ and $x_d = v \in T^*$.

A *caterpillar* graph is a tree in which all the vertices are within distance 1 of a central path. In the graph G , let C be the connected component of $G - (J^* \cup T^*)$ containing the internal vertices of P^* . We have the following lemma that helps us to prove Condition 3.

Lemma 4.4.3. *The graph C is a caterpillar with the central path being P^* . Furthermore, the only vertices of C adjacent to $J^* \cup T^*$ are x_1 and x_{d-1} which are only adjacent to x_0 and x_d respectively.*

Proof. We first look at the neighborhood vertices of the path P^* in the connected compo-

ment C . Let w be such a vertex which is adjacent to a vertex x_i with $i \in \{1, 2, \dots, d-1\}$. We prove that w is not adjacent to any other vertex in G . Thus there are no cycles in C and all the vertices in C are at distance at most 1 to P^* proving that C is a caterpillar.

We go over the possibilities of edges from w to other vertices.

Case 1: Suppose w is adjacent to some vertex x_j with $j \in \{0, \dots, d\}$. If $j = i+1$ or $j = i-1$, then wx_ix_j forms a triangle T' . Since the path $P' = u, x_1, \dots, x_i$ has length smaller than P^* and connects J^* and T' , we have that $d_G(J^*, T') < d_G(J^*, T^*)$. This contradicts the fact that (J^*, T^*) is a pair of long-claw and triangle that is the closest.

Hence w is not adjacent to x_{i-1} and x_{i+1} . Suppose $j = i+2$ or $j = i-2$. Then the graph induced by the set of vertices $\{w, x_i, x_{i+1}, x_{i+2}\}$ or $\{w, x_i, x_{i-1}, x_{i-2}\}$ is C_4 contradicting that the graph is \mathcal{G}_1 -free. Hence w is not adjacent to x_{i-2} and x_{i+2} .

Suppose w is adjacent to x_j with $1 \leq j < i-2$ or $i+2 < j \leq d$. Then note that we have a path from x_j to x_i of length 2 via w which is shorter than the path $x_i \dots x_j$ along P^* . Hence we have a path $P' = ux_1 \dots x_j wx_i \dots x_d v$ or $P' = ux_1 \dots x_i wx_j \dots x_d v$ from J^* to T^* of length smaller than P^* contradicting that P^* is the shortest such path.

Hence w is not adjacent to any of the vertices in P^* .

Case 2: Suppose w adjacent to a vertex $u' \in J^*$. We assume without loss of generality that among all neighbors of w in J^* , u' is the vertex that is closest to u in J^* , i.e, $d_{J^*}(u, u')$ is minimum. If $3 \leq i \leq d-1$, we have a path from u' to x_i of length 2 via w which is shorter than the path from u to x_i via P^* . This contradicts that P^* is the shortest path from J^* to T^* .

Suppose $i = 1$ or $i = 2$. Let P' denote a shortest path between u and u' in J^* . Since J^* is a long-claw, the length of P' is at most 4. Let us concatenate P' with the prefix of the path P^* from u to x_i which is of length at most 2. Then we get a shortest path from u' to x_i which is of length at least 2 and at most 6. The vertex w is adjacent to

only u' and x_i in this path. Hence the graph induced by the set of vertices in this path plus w is cycle C_j with $4 \leq j \leq 8$ contradicting that the graph is \mathcal{G}_1 -free.

Hence w is not adjacent to any of the vertices in J^* .

Case 3: Suppose w adjacent to a vertex $v' \in T^*$. We have $d_{T^*}(v, v') = 1$ as T^* is a triangle. If $1 \leq i \leq d - 3$, we have a path from v' to x_i of length 2 via w which is shorter than the path from v to x_i via P^* . This contradicts that P^* is the shortest path from J^* to T^* .

Hence $d - 2 \leq i \leq d - 1$. Let P' be the suffix of the path P^* from x_i to v . Then we get a shortest path from x_i to v' as $x_i P' v v'$ which is of length at least 2 and at most 3. The vertex w is adjacent only vertices x_i and v' in this path. Hence the graph induced by the set of vertices of this path plus w forms a cycle C_j with $4 \leq j \leq 5$ contradicting that the graph is \mathcal{G}_1 -free.

Hence from the above three cases, w is adjacent to none of the other vertices of $J^* \cup T^* \cup P^*$.

Case 4: We now prove that w is not adjacent to any other vertex $w' \in V(C) \setminus P^*$. Suppose that there exists such a vertex w' . We now look at various cases of the adjacency of w' with other vertices.

- **Case 4.1:** We first look at adjacencies of w' with vertices in P^* .

Suppose w' is adjacent to vertex x_i . Then the graph induced by the set of vertices $T' = \{w', w, x_i\}$ forms a triangle. Since the path $P' := u, x_1, \dots, x_i$ has length smaller than P^* and connects J^* and T' , we have that $d_G(J^*, T') < d_G(J^*, T^*)$. This contradicts the fact that (J^*, T^*) is a pair of long-claw and triangle that is the closest.

Suppose w' is adjacent to x_{i+1} or x_{i-1} . Then the graph induced by the set of vertices $\{w', w, x_i, x_{i+1}\}$ or $\{w', w, x_i, x_{i-1}\}$ forms a C_4 contradicting that the graph is \mathcal{G}_1 -free.

Hence this is not the case. Now suppose w' is adjacent to x_{i+2} or x_{i-2} . Then the graph induced by the set of vertices $\{w', w, x_i, x_{i+1}, x_{i+2}\}$ or $\{w', w, x_i, x_{i-1}, x_{i-2}\}$ forms a C_5 contradicting that the graph is \mathcal{G}_1 -free.

Hence this is also not the case. Suppose w' is adjacent to x_{i+3} or x_{i-3} . Then the graph induced by the set of vertices $\{w', w, x_i, x_{i+1}, x_{i+2}, x_{i+3}\}$ or $\{w', w, x_i, x_{i-1}, x_{i-2}, x_{i-3}\}$ forms a C_6 contradicting that the graph is \mathcal{G}_1 -free.

In the remaining case, w' is adjacent to x_j with $1 \leq j < i+3$ or $i+3 < j \leq d$. Hence $|j - i| > 3$. Then note that the path x_j, w', w, x_i is of length three which is shorter than the path between the path between x_i and x_j in P^* . Hence we get a path P' from u to v of length smaller than P^* contradicting that P^* is the smallest such path. Hence w' is not adjacent to any of the vertices in P^* .

- **Case 4.2:** Suppose w' adjacent to a vertex $u' \in J^*$. We assume without loss of generality that among all neighbors of w in J^* , u' is the vertex that is closest to u in J^* , i.e, $d_{J^*}(u, u')$ is minimum. If $3 \leq i \leq d - 1$, observe that the path $u'w'wx_i$ from u' to x_i is of length 3 which is shorter than the path from u to x_i via P^* . This contradicts that P^* is the shortest path from J^* to T^* . Hence $1 \leq i \leq 3$. Let P' denote the path between u and u' in J^* and P'' be the prefix of the path P^* from u to x_i . Then $uP''x_iww'u'P'u$ forms a cycle C_j with $4 \leq j \leq 10$ with no chords contradicting that the graph is \mathcal{G}_1 -free.

Hence w is not adjacent to any of the vertices in J^* .

- **Case 4.3:** Suppose w' adjacent to a vertex $v' \in T^*$. We have $d_{T^*}(v, v') = 1$ as T^* is a triangle. If $1 \leq i \leq d - 4$, we have the path v', w', w, x_i from v' to x_i of length 3 which is shorter than the path from v to x_i via P^* . This contradicts that P^* is the shortest path from J^* to T^* . Hence $d - 3 \leq i \leq d - 1$. Let P' be the suffix of the path P^* from x_i to v . Then $v'w'wx_iP'vv'$ forms a cycle C_j with $4 \leq j \leq 6$ without any chords contradicting that the graph is \mathcal{G}_1 -free.

Hence we conclude that w' is not adjacent to any of the vertices in $J^* \cup T^* \cup P^*$. Now look

the graph induced by the set of vertices J' which is

- $\{w', w, x_i, x_{i-1}, x_{i-2}, x_{i+1}, x_{i+2}\}$ for $3 \leq i \leq x_{d-2}$ or
- $\{w', w, x_i, x_{i-1}, u, x_{i+1}, x_{i+2}\}$ when $i = 2$ or
- $\{w', w, x_i, u, u', x_{i+1}, x_{i+2}\}$ when $i = 1$ for $u' \in J^* \cap N(u)$ or
- $\{w', w, x_i, x_{i-1}, x_{i-2}, v, v'\}$ when $i = d - 1$ for $v' \in J^* \cap N(v)$.

In all cases, the graph induced by J' forms a long-claw. Since the path P' from J' to $v \in T^*$ has length smaller than P^* , we have that $d_G(J', T^*) < d_G(J^*, T^*)$. This contradicts the fact that (J^*, T^*) is a pair of long-claw and triangle that is closest.

Hence no such vertex w' exists and therefore w has no other neighbors in G .

Hence, the graph C is a caterpillar with the central path being P^* . Furthermore, no vertices other than x_1 and x_{d-1} is adjacent to $J^* \cup T^*$. \square

We now use Lemma 4.4.3 to prove that Condition 3 is satisfied for INTERVAL-OR-TREE DELETION.

Lemma 4.4.4. *Condition 3 is satisfied for INTERVAL-OR-TREE DELETION.*

Proof. Condition 3 is not satisfied in the following case. There exist a pair (J^*, T^*) which is the vertex subsets of a closest long-claw, triangle pair in a connected component of G , where J^* is a long-claw and T^* is a triangle. Also there is a shortest path $P^* := x_0, x_1, \dots, x_{d-1}, x_d$ between J^* and T^* of length $d_G(J^*, T^*) = d$ with $x_0 = u \in J^*$ and $x_d = v \in T^*$. A forbidden set Q of the graph G is such that Q contains some internal vertex x_i of the path P but it does not contain v .

Since the graph G is \mathcal{G}_1 -free, $G[Q]$ can be one of hole, \dagger -AW or a \ddagger -AW or contain a forbidden pair for (long claw, triangle). Note that all of these possibilities contain cycles. But from Lemma 4.4.3, the component C of $G \setminus (J^* \cup T^*)$ that contains the internal vertices

of P^* is a caterpillar which does not contain any cycles. Hence Q is not fully contained in C .

From Lemma 4.4.3, the only neighbors of C is u and v via x_1 and x_{d-1} respectively. Since $G[Q]$ is connected and intersects x_i , $Q \cap \{u, v\} \neq \emptyset$. Since we assumed that Q does not contain the vertex v , we have $u \in Q$. In particular, Q contains the entire subpath of P^* from u to x_i .

Also note that u cannot be part of a subset of three vertices T' which is a triangle as otherwise, we get a pair (J^*, T') with distance zero contradicting that (J^*, T^*) was the closest pair P^* has internal vertices. Hence the forbidden set Q cannot be \dagger -AW or a \ddagger -AW whose structure forces u to be part of a triangle if it contains x_i (which also happens only in the case when $i = 1$). Since x_i does not have any paths to the vertex u other than the subpath in P^* , the forbidden set Q cannot be a hole as well.

Hence Q can only correspond to a (long claw, triangle) forbidden pair. In this case, we claim that the set after removing the vertices x_1, \dots, x_i from Q is also a forbidden set. This contradicts that Q is a forbidden set as by definition they are required to be minimal.

Since Q is a forbidden set, it contains vertex subsets that are isomorphic to a long-claw and a triangle. From Lemma 4.4.3, we can conclude that none of the vertices x_1, \dots, x_i can be part of any triangle in G . None of these vertices can be part of a long-claw in G either as it contradicts that (J^*, T^*) is the closest forbidden pair. Since v disconnects the path P^* , the subpath x_1, \dots, x_i is not part of a path connecting a long-claw and a triangle either as if so Q must contain the entire path P^* including v . Hence the set after removing the vertices x_1, \dots, x_i from Q is still a forbidden set contradicting that Q is minimal.

These cases of Q are mutually exhaustive completing the proof of the Lemma. □

Hence, we have established that INTERVAL-OR-TREE DELETION is indeed an example of SPECIAL INFINITE- (Π_1, Π_2) -DELETION. We have the following theorem.

Theorem 18. INTERVAL-OR-TREE DELETION *has an FPT algorithm with running time*

$\mathcal{O}^*(10^k)$ and a 10-approximation algorithm.

Proof. We have $f(k) = \max\{10^k, 3.618^k\} = 10^k$ and $c = \max_{(H_1, H_2) \in \mathcal{F}_p} (|H_1| + |H_2|) = 10$. Hence from Theorem 16, we have a $\mathcal{O}^*(10^k)$ time algorithm for INTERVAL-OR-TREE DELETION.

We know that INTERVAL VERTEX DELETION has an 10-approximation algorithm [34] and FEEDBACK VERTEX SET has a 2-approximation algorithm [11]. Hence $d = \max\{c, c_1, c_2\} = 10$ giving a 10-approximation for INTERVAL-OR-TREE DELETION from Theorem 17. \square

4.4.2 Proper Interval or Trees

We define the problem.

PROPER INTERVAL-OR-TREE DELETION

Input: An undirected graph $G = (V, E)$ and an integer k

Parameter: k

Question: Is there $S \subseteq V(G)$ of size at most k such that every connected component of $G - S$ is a proper interval graph or a tree?

We have the following forbidden subgraph characterization of proper interval graphs.

Lemma 4.4.5. [28] *A graph is said to be a proper interval graph if and only if it does not contain claw, net, sun or hole as its induced subgraphs.*

We now give a characterization for graphs whose every connected component is a proper interval graph or a tree.

Lemma 4.4.6. *The following statements are equivalent.*

1. *A graph G is such that every connected component of G is a proper interval graph or a tree.*

2. A graph G does not have any net, sun or hole as its induced subgraphs. Moreover, no connected component of G have a claw and a triangle as induced graphs.

Proof. We prove that the forbidden pair family is (claw, triangle). The forbidden family \mathcal{F}_1 for proper interval graphs are claw, net, sun or holes. The forbidden family of trees \mathcal{F}_2 is cycles. Since cycles of length at least 4 are common in \mathcal{F}_1 and \mathcal{F}_2 , they are in $\text{sp}(\mathcal{F}_1, \mathcal{F}_2)$. Since the graphs net and sun have triangle as induced subgraph, they are in $\text{sp}(\mathcal{F}_1, \mathcal{F}_2)$ as well. The only remaining pair in $\mathcal{F}_1 \times \mathcal{F}_2$ is (claw, triangle). The proof now follows from the forbidden characterization in Lemma 4.3.2. \square

Hence Condition 2 is satisfied by PROPER INTERVAL-OR-TREE DELETION as the forbidden pair is of size one which is (claw, triangle). Since PROPER INTERVAL VERTEX DELETION is FPT with a $O^*(6^k)$ running time algorithm from [152] and FEEDBACK VERTEX SET is FPT with a $O^*(3.618^k)$ running time algorithm from [106], Condition 1 is satisfied as well.

We now prove that Condition 3 is satisfied by PROPER INTERVAL-OR-TREE DELETION. Recall Lemma 4.3.5 where we established that the internal vertices of any shortest path between the vertex sets of a closest claw, triangle pair in a graph do not contain any neighbors other than the endpoints of the path.

Lemma 4.4.7. *Condition 3 is satisfied by PROPER INTERVAL-OR-TREE DELETION.*

Proof. Condition 3 is not satisfied in the following case. There exist a pair (J^*, T^*) which is the vertex subsets of a closest claw, triangle pair in a connected component of G , where J^* is a claw and T^* is a triangle. Also there is a shortest path $P^* := x_0, x_1, \dots, x_{d-1}, x_d$ between J^* and T^* of length $d_G(J^*, T^*) = d$ with $x_0 = u \in J^*$ and $x_d = v \in T^*$. A forbidden set Q of the graph G is such that Q contains some internal vertex x_i of the path P but it does not contain v .

Let $\mathcal{G}_1 = \emptyset$. The graph $G[Q]$ can be one of net, sun, hole or contain a forbidden pair

for (claw, triangle). Note that all of these possibilities contain cycles. But from Lemma 4.3.5, the component C of $G \setminus (J^* \cup T^*)$ that contains the internal vertices of P^* is the path $P^* \setminus \{u, v\}$ which does not contain any cycles. Hence Q is not fully contained in C .

From Lemma 4.3.5, the only neighbors of C is u and v via x_1 and x_{d-1} respectively. Since $G[Q]$ is connected and intersects x_i , $Q \cap \{u, v\} \neq \emptyset$. Since we assumed that Q does not contain the vertex v , we have $u \in Q$. In particular, Q contains the entire subpath of P^* from u to x_i .

Also note that u cannot be part of a subset of three vertices T' which is a triangle as otherwise, we get a pair (J^*, T') with distance zero contradicting that (J^*, T^*) was the closest pair P^* has internal vertices. Hence the forbidden set Q cannot be a net or a sun whose structure forces u to be part of a triangle if it contains x_i . Since x_i does not have any paths to the vertex u other than the subpath in P^* , the forbidden set Q cannot be a hole as well.

Hence Q can only correspond to a (claw, triangle) forbidden pair. In this case, we claim that the set after removing the vertices x_1, \dots, x_i from Q is also a forbidden set. This contradicts that Q is a forbidden set as by definition they are required to be minimal.

Since Q is a forbidden set, it contains vertex subsets that are isomorphic to a claw and a triangle. From Lemma 4.3.5, we can conclude that none of the vertices x_1, \dots, x_i can be part of any triangle in G . None of these vertices can be part of a claw in G either as it contradicts that (J^*, T^*) is the closest forbidden pair. Since v disconnects the path P^* , the subpath x_1, \dots, x_i is not part of a path connecting a claw and a triangle either as if so Q must contain the entire path P^* including v . Hence the set after removing the vertices x_1, \dots, x_i from Q is still a forbidden set contradicting that Q is minimal.

These cases of Q are mutually exhaustive completing the proof of the Lemma. □

We have $f(k) = \max\{6^k, 3.618^k\} = 6^k$ and $c = 7$. Also we know that PROPER INTERVAL VERTEX DELETION has a 6-approximation algorithm [152] and FEEDBACK VERTEX

SET has a 2-approximation algorithm [11]. Hence we have $d = 7$ as well. We have the following theorem.

Theorem 19. PROPER INTERVAL-OR-TREE DELETION *can be solved in $\mathcal{O}^*(7^k)$ -time and has a 7-approximation algorithm.*

4.4.3 Chordal or Bipartite Permutation

We define the problem as follows.

CHORDAL-OR-BIPARTITE PERMUTATION DELETION

Input: An undirected graph $G = (V, E)$ and an integer k

Parameter: k

Question: Is there $S \subseteq V(G)$ of size at most k such that every connected component of $G - S$ is either a chordal graph, or a bipartite permutation graph?

The forbidden set for chordal graphs \mathcal{F}_1 is the set of cycle graphs with a length of at least 4.

We have the following characterization for bipartite permutation graphs which defines \mathcal{F}_2 .

Lemma 4.4.8. ([26]) *A graph is said to be a bipartite permutation graph if and only if it does not contain long-claw, X_2, X_3, C_3 or cycle graphs of length at least 5 as its induced subgraphs. See Figure 4.2 for an illustration of the graphs X_2 and X_3 .*

We now give a characterization for graphs whose each connected component is either a chordal graph or a bipartite permutation graph.

Lemma 4.4.9. *The followings are equivalent.*

1. *Let G be a graph such that every connected component is either chordal or a bipartite permutation graph.*
2. *G does not have any X_2, X_3 or induced cycle of length at least 5 as induced subgraphs. Moreover, G cannot have long-claw and C_4 in the same connected component or*

have C_4 and triangle in the same connected component.

Proof. We prove that the forbidden pair family is $(C_4, \text{long-claw})$ and (C_4, C_3) . The forbidden family \mathcal{F}_1 for chordal graphs are holes. The forbidden family of bipartite permutation graphs \mathcal{F}_2 is long-claw, X_2, X_3, C_3 plus cycles of length at least 5. Since cycles of length at least 5 are common in \mathcal{F}_1 and \mathcal{F}_2 , they are in $\text{sp}(\mathcal{F}_1, \mathcal{F}_2)$. Since the graphs X_2, X_3 have C_4 as induced subgraph, they are in $\text{sp}(\mathcal{F}_1, \mathcal{F}_2)$ as well. The only remaining pairs in $\mathcal{F}_1 \times \mathcal{F}_2$ are $(C_4, \text{long-claw})$ and $(C_4, \text{triangle})$. The proof now follows from the forbidden characterization in Lemma 4.3.2. \square

Hence Condition 2 is satisfied by CHORDAL-OR-BIPARTITE PERMUTATION DELETION as the forbidden pair is of size two which are $(\text{long-claw}, C_4)$ and $(\text{triangle}, C_4)$. Since CHORDAL VERTEX DELETION is FPT with a $\mathcal{O}^*(k^{\mathcal{O}(k)})$ running time algorithm from [35] and BIPARTITE PERMUTATION VERTEX DELETION SET is FPT with a $\mathcal{O}^*(9^k)$ running time algorithm from [26], Condition 1 is satisfied as well.

It remains to show that Condition 3 is satisfied by CHORDAL-OR-BIPARTITE PERMUTATION DELETION. We define \mathcal{G}_1 as all the forbidden graphs in $\text{sp}(\mathcal{F}_1, \mathcal{F}_2)$ of size at most 10.

Let (J^*, T^*) be the vertex subsets of a closest forbidden pair in a connected component of a \mathcal{G}_1 -free G , where J^* is one of long-claw or triangle and T^* is a C_4 . Let $P^* := x_0, x_1, \dots, x_{d-1}, x_d$ be a shortest path between J^* and T^* of length $d_G(J^*, T^*) = d$ with $x_0 = u \in J^*$ and $x_d = v \in T^*$.

Let C be the connected component of $G - (J^* \cup T^*)$ containing the internal vertices of P^* . We have the following lemma similar to Lemma 4.4.3 in INTERVAL-OR-TREE DELETION.

Lemma 4.4.10. *The graph C is a caterpillar with the central path being $P^* \setminus \{u, v\}$. Furthermore, the only vertices of C adjacent to $J^* \cup T^*$ are x_1 and x_{d-1} which are only adjacent to x_0 and x_d respectively.*

Proof. Let w be a vertex of C other than P^* adjacent to a vertex x_i with $i \in [d-1]$. We claim that w is not adjacent to any other vertex in G .

We go over possibilities of edges from w to other vertices.

Case 1: Suppose w is adjacent to some vertex x_j with $j \in \{0, \dots, d\}$. If $j = i+1$ or $j = i-1$, then wx_ix_j forms a triangle J' . Since the path $P' = x_i, \dots, x_{d-1}, v$ that has length smaller than P^* connects J' and T^* , we have that $d_G(J', T^*) < d_G(J^*, T^*)$. This contradicts the fact that (J^*, T^*) is a closest forbidden pair.

Hence w is not adjacent to x_{i-1} and x_{i+1} . Suppose $j = i+2$ or $j = i-2$. Then the graph induced by the vertices $\{w, x_i, x_{i+1}, x_{i+2}\}$ or $\{w, x_i, x_{i-1}, x_{i-2}\}$ forms the graph T' which is a C_4 . Since the path $P' = u, \dots, x_i$ that has length smaller than P^* connects J^* and T' , we have that $d_G(J^*, T') < d_G(J^*, T^*)$. This contradicts the fact that (J^*, T^*) is a closest forbidden pair.

Suppose now w is adjacent to x_j with $0 \leq j < i-2$ or $i+2 < j \leq d$. Then note that we have a path from x_j to x_i of length two via w which is shorter than the path $x_i \dots x_j$ via P^* . This creates a path P' which is one of $u, x_1, \dots, x_j, w, x_i, \dots, x_d, v$ or $u, x_1, \dots, x_i, w, x_j, \dots, x_d, v$ from J^* to T^* of length smaller than P^* contradicting that P^* is the shortest such path. Hence w is not adjacent to any of the vertices in P^* .

Case 2: Suppose w adjacent to a vertex $u' \in J^*$. We assume without loss of generality that among all neighbors of w in J^* , u' is the vertex that is closest to u in J^* , i.e. $d_{J^*}(u, u')$ is minimum. We have $d_{J^*}(u, u') \leq 4$ as J^* is either a long-claw or a triangle. If $3 \leq i \leq d-1$, we have a path from u' to x_i of length 2 via w which is shorter than the path from u to x_i via P^* . This contradicts that P^* is the shortest path from J^* to T^* . Hence $i = 1$ or $i = 2$. Let P' denote the path between u and u' in J^* and P'' be the subpath of P^* from u to x_i . Then $u, P''x_iwu'P'u$ forms a cycle C_j with $4 \leq j \leq 8$ without any chords. This either contradicts that (J^*, T^*) is a closest forbidden pair or G is \mathcal{G}_1 -free. Hence w is not adjacent to any of the vertices in J^* .

Case 3: Suppose w adjacent to a vertex $v' \in T^*$. We have $d_{T^*}(v, v') \leq 2$ as T^* is a C_4 . If $1 \leq i \leq d-3$, we have a path from v' to x_i of length 2 via w which is shorter than the path from v to x_i via P^* . This contradicts that P^* is the shortest path from J^* to T^* . Hence $d-2 \leq i \leq d-1$. Let P' be the subpath of P^* from x_i to v . Then $v'wx_iP'vv'$ forms a cycle C_j with $4 \leq j \leq 6$ without any chords. This either contradicts that (J^*, T^*) is a closest forbidden pair or G is \mathcal{G}_1 -free.

Hence w is adjacent to none of the other vertices of $J^* \cup T^* \cup P^*$.

Case 4: We now prove that w is not adjacent to any other vertex $w' \in V(G) - (J^* \cup T^* \cup P^*)$.

Suppose that there exists such a vertex w' . Suppose w' is adjacent to vertex x_i . Then the graph induced by the set of vertices $J' = \{w', w, x_i\}$ forms a triangle. Since the path $P' = x_i, \dots, x_{d-1}, v$ that has length smaller than P^* connects J' and T^* , we have that $d_G(J', T^*) < d_G(J^*, T^*)$. This contradicts the fact that (J^*, T^*) is a closest forbidden pair.

- **Case 4.1:** Suppose w' is adjacent to x_{i+1} or x_{i-1} . Then the graph T' induced by the vertices $\{w', w, x_i, x_{i+1}\}$ or $\{w', w, x_i, x_{i-1}\}$ forms a C_4 . Since the path $P' = u, \dots, x_i$ that has length smaller than P^* connects J^* and T' , we have that $d_G(J^*, T') < d_G(J^*, T^*)$. This contradicts the fact that (J^*, T^*) is a closest forbidden pair.

Hence this is not the case. Now suppose w' is adjacent to x_{i+2} or x_{i-2} . Then the graph induced by the set of vertices $\{w', w, x_i, x_{i+1}, x_{i+2}\}$ or $\{w', w, x_i, x_{i-1}, x_{i-2}\}$ is C_5 contradicting that G is \mathcal{G}_1 -free. Hence this is not the case. Now suppose w' is adjacent to x_{i+3} or x_{i-3} . Then the graph induced by the set of vertices $\{w', w, x_i, x_{i+1}, x_{i+2}, x_{i+3}\}$ or $\{w', w, x_i, x_{i-1}, x_{i-2}, x_{i-3}\}$ is C_6 contradicting that G is \mathcal{G}_1 -free.

Now suppose w' is adjacent to x_j with $1 \leq j < i+3$ or $i+3 < j \leq d$. Then we have a path P' which is either $u, x_1 \dots x_j w, x_i \dots x_d, v$ or $u, x_1 \dots x_i w, x_j \dots x_d, v$ from J^* to T^* of length smaller than P^* contradicting that P^* is the smallest such path. Hence w' is not adjacent to any of the vertices in P^* .

- **Case 4.2:** Suppose w' adjacent to a vertex $u' \in J^*$. We assume without loss of generality

that among all neighbors of w in J^* , u' is the vertex that is closest to u in J^* , i.e., $d_{J^*}(u, u')$ is minimum. If $3 \leq i \leq d-1$, we have the path $u'w'wx_i$ from u' to x_i of length 3 via w which is shorter than the path from u to x_i via P^* . This contradicts that P^* is the shortest path from J^* to T^* . Hence $1 \leq i \leq 3$. Let P' denote the between u and u' in J^* and P'' be the prefix of the path P^* from u to x_i . Then $uP''x_i, w, w', u'P'u$ forms a cycle C_j with $4 \leq j \leq 10$ with no chords. This either contradicts that (J^*, T^*) is a pair that is closest or G is \mathcal{G}_1 -free. Hence w is not adjacent to any of the vertices in J^* .

- **Case 4.3:** Suppose w' adjacent to a vertex $v' \in T^*$. We have $d_{T^*}(v, v') \leq 2$ as T^* is a C_4 . If $1 \leq i \leq d-4$, we have the path v', w', w, x_i from v' to x_i of length three which is shorter than the path from v to x_i via P^* . This contradicts that P^* is the shortest path from J^* to T^* . Hence $d-3 \leq i \leq d-1$. Let P' be the suffix of the path P^* from x_i to v . Then $v', w', w, x_iP'v, v'$ forms a cycle C_j with $4 \leq j \leq 7$. This either contradicts that (J^*, T^*) is a pair that is closest or G is \mathcal{G}_1 -free.

Hence we conclude that w' is not adjacent to any of the vertices in $J^* \cup T^* \cup P^*$. Now, we look at the induced subgraph formed by the set of vertices J' which is

- $\{w', w, x_i, x_{i-1}, x_{i-2}, x_{i+1}, x_{i+2}\}$ for $3 \leq i \leq x_{d-2}$,
- $\{w', w, x_i, x_{i-1}, u, x_{i+1}, x_{i+2}\}$ when $i = 2$,
- $\{w', w, x_i, u, u', x_{i+1}, x_{i+2}\}$ when $i = 1$ for $u \in J^* \cap N(u)$ or
- $\{w', w, x_i, x_{i-1}, x_{i-2}, v, v'\}$ when $i = d-1$ for $v' \in J^* \cap N(v)$.

This graph forms a long-claw. Since the path P' from J' to $v \in T^*$ has length smaller than P^* , the distance $d_G(J', T^*) < d_G(J^*, T^*)$. This contradicts the fact that (J^*, T^*) is a closest forbidden pair.

Hence no such vertex w' exist and therefore w has no other neighbors in G . □

We now use Lemma 4.4.10 to prove that Condition 3 is satisfied for CHORDAL-OR-BIPARTITE PERMUTATION DELETION.

Lemma 4.4.11. *Condition 3 is satisfied for CHORDAL-OR-BIPARTITE PERMUTATION DELETION.*

Proof. Condition 3 is not satisfied in the following case. There exist a pair (J^*, T^*) which is the vertex subsets of a closest long-claw, triangle pair in a connected component of G , where J^* one of long-claw or a triangle and T^* is a C_4 . Also there is a shortest path $P^* := x_0, x_1, \dots, x_{d-1}, x_d$ between J^* and T^* of length $d_G(J^*, T^*) = d$ with $x_0 = u \in J^*$ and $x_d = v \in T^*$. A forbidden set Q of the graph G is such that Q contains some internal vertex x_i of the path P but it does not contain v .

Since the graph G is \mathcal{G}_1 -free, $G[Q]$ can be a hole of size at least 11 or contain a forbidden pair which is (long claw, C_4) or (C_3, C_4) . Note that all of these possibilities contain cycles. But from Lemma 4.4.10, the component C of $G \setminus (J^* \cup T^*)$ that contains the internal vertices of P^* is a caterpillar which does not contain any cycles. Hence Q is not fully contained in C .

From Lemma 4.4.10, the only neighbors of C is u and v via x_1 and x_{d-1} respectively. Since $G[Q]$ is connected and intersects x_i , $Q \cap \{u, v\} \neq \emptyset$. Since we assumed that Q does not contain the vertex v , we have $u \in Q$. In particular, Q contains the entire subpath of P^* from u to x_i .

Since x_i does not have any paths to the vertex u other than the subpath in P^* , the forbidden set Q cannot be a hole. Hence Q can only correspond to a (long claw, C_4) or (C_3, C_4) forbidden pair. In this case, we claim that the set after removing the vertices x_1, \dots, x_i from Q is also a forbidden set. This contradicts that Q is a forbidden set as by definition they are required to be minimal.

From Lemma 4.4.3, we can conclude that none of the vertices x_1, \dots, x_i can belong to a subset of vertices in the graph such that the graph induced by the subset is a long-claw,

triangle or a C_4 . This is because otherwise, we get a forbidden pair using this subset which is closer than the closest forbidden pair (J^*, T^*) . Since v disconnects the path P^* , the subpath x_1, \dots, x_i is not part of a path connecting a forbidden pair either as if so Q must contain the entire path P^* including v . Hence the set after removing the vertices x_1, \dots, x_i from Q is still a forbidden set contradicting that Q is minimal.

These cases of Q are mutually exhaustive completing the proof of the Lemma. \square

We have $f(k) = \max\{k^{O(k)}, 9^k\} = k^{O(k)}$ and $c = 11$. We know that CHORDAL VERTEX DELETION has an $\log^2(OPT)$ -approximation algorithm [3] where OPT denote the size of the optimal solution. Also, BIPARTITE PERMUTATION VERTEX DELETION has a 9-approximation algorithm [26]. Hence $d = \max\{c, c_1, c_2\} = \max\{11, \log^2(OPT), 2\} = \log^2(OPT)$. We have the following theorem.

Theorem 20. CHORDAL-OR-BIPARTITE PERMUTATION DELETION *can be solved in $k^{O(k)}$ poly(n)-time and has as a $\log^2(OPT)$ -approximation algorithm..*

4.5 Conclusion

We gave faster algorithms for some vertex deletion problems to pairs of scattered graph classes with infinite forbidden families.

Other than the problems mentioned in the Conclusion section of Chapter 3, an open problem is to give faster FPT algorithms for problems that do not fit in any of the frameworks described above, especially problems which do not have a constant sized forbidden pair family. An example is the case when (Π_1, Π_2) is (Chordal, Bipartite). The forbidden pair family for this problem is the set of all pairs (C_{2i}, C_3) with $i \geq 2$ which is not of constant size.

Part II

Deletion distance parameterizations

Chapter 5

Structural Parameterizations with Modulator Oblivion

In this chapter, we look at several problems parameterized by deletion distance to chordal graphs. Specifically, we look at VERTEX COVER, FEEDBACK VERTEX SET and ODD CYCLE TRANSVERSAL and some generalizations of these problems, parameterized by the size of a chordal vertex deletion set (CVD), as these problems are polynomial time solvable in chordal graphs [80, 148, 41].

In problems for which the parameter is the size of a modulator, it is also assumed that the modulator is given with the input. This assumption can be removed if finding the modulator is also FPT parameterized by the modulator size. However, there are instances where finding the modulator is more expensive than solving the problem if the modulator is given. For example, finding a subset of k vertices whose deletion results in a perfect graph is known to be $W[2]$ -hard [84]. But if the deletion set is given, then (as seen in Chapter 1) VERTEX COVER is FPT when parameterized by the size of the deletion set.

Hence Fellows et al. [64] ask whether VERTEX COVER is FPT when parameterized by a (promised) bound on the vertex-deletion distance to a perfect graph, without giving a minimum deletion set in the input. While we do not answer this question, we address a

similar question in the context of problems parameterized by the distance to chordal graphs, another well-studied class of graphs where VERTEX COVER is polynomial time solvable whereas the best-known algorithm to find a k -sized chordal deletion set takes $k^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ time [35]. We also do not know of a constant factor (FPT) approximation algorithm for CVD even with $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ running time. There are many recent results on polynomial time approximation algorithms for CHORDAL VERTEX DELETION [91, 3, 101] with the current best algorithm having a $\mathcal{O}(\log^2 \text{opt})$ ratio, where opt is the size of minimum CVD [3]. If we use this approximation algorithm and do branching (see Related Work in this section), then we can obtain a $2^{\mathcal{O}(k \log^2 k)}n^{\mathcal{O}(1)}$ time algorithm for VERTEX COVER.

Hence, in a similar vein to the question by Fellows et al., we ask whether (minimum) VERTEX COVER (and other related problems) can be solved in $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ time with only a promise on the size k of the chordal deletion set, and answer the question affirmatively. Our algorithms even go one step further, in not even needing the promise. They solve the problem or determine that the chordal deletion set is of size more than k .

Our Results: Specifically, we give $\mathcal{O}^*(2^{\mathcal{O}(k)})$ algorithms for the problems defined below.

d-COLORABLE SUBGRAPH BY CVD

Input: A graph $G = (V, E)$ and $k, \ell, d \in \mathbb{N}$.

Parameter: k

Question: Determine if there is a vertex set X of size at most ℓ in G such that $G - X$ is d -colorable or output that minimum chordal vertex deletion set of G is of size more than k ?

When $d = 1$ and $d = 2$, the problem reduces to VERTEX COVER BY CVD and ODD CYCLE TRANSVERSAL BY CVD where we require the graph $G - X$ to be an independent set and bipartite, respectively. We also define FEEDBACK VERTEX SET BY CVD where we require the graph $G - X$ to be a forest.

We remark that our algorithms do not necessarily address the question of whether the input

graph has a CVD of size at most k , and may solve the problem sometimes even when the CVD size is more than k .

We also show that all the problems mentioned above cannot be solved in $\mathcal{O}^*((2 - \varepsilon)^k)$ time under Strong Exponential Time Hypothesis (SETH) even if a CVD of size k is given as part of the input. This matches the upper bound of the known algorithm for VERTEX COVER BY CVD when the modulator is given.

Related Work:

When CVD is given: If we are given a CVD S of size k along with an n -vertex graph G as the input, then we have a $2^k n^{\mathcal{O}(1)}$ time algorithm (call it \mathcal{A}) for VERTEX COVER as chordal graphs are hereditary. An FPT algorithm with $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ time for FEEDBACK VERTEX SET BY CVD is given by Jansen et al [97] where they first find the modulator. This algorithm follows the algorithm to find a minimum feedback vertex set in bounded treewidth graphs. A similar trick works for ODD CYCLE TRANSVERSAL too, when the modulator is given.

When the modulator is given, the FPT algorithms discussed above have been generalized for other problems and other classes of graphs (besides those that are k away from the class of chordal graphs). Let Φ be a Counting Monadic Second Order Logic (CMSO) formula and $t \geq 0$ be an integer. For a given graph $G = (V, E)$, the task is to maximize $|X|$ subject to the following constraints: there is a set $F \subseteq V$ such that $X \subseteq F$, the subgraph $G[F]$ induced by F is of treewidth at most t , and structure $(G[F], X)$ models Φ . Note that the problem corresponds to finding a minimum vertex cover and a minimum feedback vertex set when $t = 0$ and $t = 1$ respectively when Φ is a tautology. For a polynomial $poly$, let G_{poly} be the class of graphs such that, for any $G \in G_{poly}$, graph G has at most $poly(n)$ minimal separators. Fomin et al [73] gave a polynomial time algorithm for solving this optimization problem on the graph class G_{poly} . Consider $G_{poly} + kv$ to be the graph class formed from G_{poly} where to each graph we add at most k vertices of arbitrary adjacencies. Liedloff et

al. [111] further proved that the above problem is FPT on $G_{poly} + kv$, with parameter k , where the modulator is also a part of the input. As a chordal graph has polynomially many minimal separators [80], we obtain that this problem parameterized by CVD size is FPT when the modulator is given.

Other ‘permissive’ problems. Similar problems have been termed as ‘permissive problems’ in the context of testing satisfiability of CSPs (constraint satisfaction problems) with small sized strong backdoors [79]. While detecting strong backdoors to a general CSP is hard, the authors address the question of satisfiability of CSPs where the backdoor set is not given, and the algorithm was supposed to solve satisfiability or determine that the backdoor set size is more than k .

An example line of work where a faster constant factor approximation algorithm is available is in the context of optimization problems parameterized by treewidth.

For example, the INDEPENDENT SET problem parameterized by treewidth of the graph tw can be solved using standard dynamic programming (DP) in $2^{tw} \cdot tw^{O(1)} \cdot n$ time [45]. But the best known algorithm for outputting a tree-decomposition of minimum width takes time $tw^{O(tw^3)} n$ [18]. Thus, the total running time is $tw^{O(tw^3)} n$, when a tree decomposition is not given as an input. But one can overcome this by obtaining a tree decomposition of width $5tw$ in time $2^{O(tw)} n$ [20] and then applying the DP algorithm over the tree decomposition.

One previous example we know of a parameterized problem where the FPT algorithm solves the problem without the modulator or even the promise, is VERTEX COVER parameterized by the size of KÖNIG VERTEX DELETION set k . A König vertex deletion set of G is a subset of vertices of G whose removal results in a graph where the size of its minimum vertex cover and maximum matching is the same. In VERTEX COVER BY KÖNIG VERTEX DELETION, we are given graph $G = (V, E)$, $k, \ell \in \mathbb{N}$ and an assumption that there exists a König vertex deletion set of size k in G , here k is parameter. We ask whether there exists a vertex cover of size ℓ in G . Lokshantov et al. [117] solve VERTEX COVER BY KÖNIG

VERTEX DELETION in $\mathcal{O}^*(1.5214^k)$ time without the promise.

Finally, we remark that there is an analogous line of work in the classical world of polynomial time algorithms. For example, it is known that finding a maximum clique in a unit disk graph is polynomial time solvable given a unit disk representation of the unit disk graph [40], though it is NP-hard to recognize whether a given graph is a unit disk graph [29]. Raghavan and Spinrad [142] give a permissive algorithm that given a graph either finds a maximum clique in the graph or outputs a certificate that the given graph is not a unit disk graph. See also [27, 82, 73] for some other examples of permissive algorithms.

Our Techniques:

The first step in our algorithms is to obtain, what we call a semi clique tree decomposition of the given graph if one exists. It is known [80] that every chordal graph has a clique-tree decomposition, i.e., a tree decomposition where every bag is a clique in the graph. If the modulator is given, then we can add it to each bag, and obtain a tree-decomposition where each bag is a clique plus at most k vertices. In our case (where the modulator is not given), we obtain a tree decomposition in $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ time where each bag can be partitioned into $C \uplus N$, where C can be covered by at most 4 cliques in G and $|N| \leq 7k + 5$. Here we also know a partition $C_1 \uplus C_2 \uplus C_3 \uplus C_4$ of C where each C_i is a clique. We call this tree decomposition a $(4, 7k + 5)$ -semi clique tree decomposition. Our result in this regard is formalized in the following theorem.

Theorem 21. *There is an algorithm that given a graph G and an integer k runs in time $\mathcal{O}(2^{7k} \cdot (kn^4 + n^{\omega+2}))$ where ω is the matrix multiplication exponent and either constructs a $(4, 7k + 5)$ -semi clique tree decomposition \mathcal{T} of G or concludes that there is no chordal vertex deletion set of size k in G . Moreover, the algorithm also provides a partition $C_1 \uplus C_2 \uplus C_3 \uplus C_4 \uplus N$ of each bag of \mathcal{T} such that $|N| \leq 7k + 5$ and C_i is a clique in G for all $i \in \{1, 2, 3, 4\}$.*

After getting a $(4, 7k + 5)$ -semi clique tree decomposition, we then design DP algorithms

for VERTEX COVER, FEEDBACK VERTEX SET and ODD CYCLE TRANSVERSAL on this tree decomposition. Since the vertex cover of a clique has to contain all but one vertex of the clique, the number of ways the solution might intersect a bag of the tree is at most $\mathcal{O}(2^{7k}n^4)$. Using this fact, one can bound the running time for the DP algorithm for VERTEX COVER BY CVD to $\mathcal{O}(2^{7k}n^5)$. The overall running time would be the sum of the time taken to construct a $(4, 7k + 5)$ -semi clique tree decomposition and the time of the DP algorithm on this tree decomposition which is bounded by $\mathcal{O}(2^{7k}n^5)$. In the case of FEEDBACK VERTEX SET and ODD CYCLE TRANSVERSAL, again from each clique all but two vertices will be in the solution. Using this fact one can bound the running time of FEEDBACK VERTEX SET BY CVD and ODD CYCLE TRANSVERSAL BY CVD to be $\mathcal{O}^*(2^{O(k)})$.

Very recently, Fomin and Golovach [65] give subexponential algorithms to various problems on graphs which can be turned into a chordal graph by adding k edges. Similar to the line of work in this chapter, they come up with an almost-clique tree decomposition (where each bag can be converted to a clique by adding k edges) and then apply dynamic programming algorithms on this tree decompositions. We use the dynamic programming algorithms in this chapter on the tree decomposition we constructed to give algorithms for d -COLORABLE SUBGRAPH parameterized by minimum CVD size.

Organization of the chapter. In Section 5.1, we give the necessary preliminaries. In Section 5.2, we prove Theorem 21. In Section 5.3, we first address d -COLORABLE SUBGRAPH BY CVD using dynamic programming on semi clique tree decomposition. We then give more direct and faster algorithms for VERTEX COVER BY CVD and ODD CYCLE TRANSVERSAL BY CVD and also for FEEDBACK VERTEX SET BY CVD. We then conclude this section with lower bounds on these problems assuming SETH.

5.1 Preliminaries

Proposition 5.1.1 ([57]). *Let G be a graph and C be a clique in G . Let $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ be a tree decomposition of G . Then, there is a node $t \in V(T)$ such that $C \subseteq X_t$.*

Definition 5.1.1 (Clique tree decomposition). *A clique tree decomposition of a graph G is a tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ where X_t is a clique in G for all $t \in V(T)$.*

Proposition 5.1.2 ([80]). *A graph is chordal if and only if it has a clique tree decomposition.*

Definition 5.1.2. *A graph G is called an (c, ℓ) -semi clique if there is a partition $C \uplus N$ of $V(G)$ such that $G[C]$ is a union of at most c cliques and $|N| \leq \ell$.*

Definition 5.1.3 ((c, ℓ) -semi clique tree decomposition). *For a graph G and $c, \ell \in \mathbb{N}$, a tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ of G is a (c, ℓ) -semi clique tree decomposition if $G[X_t]$ is a (c, ℓ) -semi clique for each $t \in V(T)$.*

We define the NODE MULTIWAY CUT problem where we are given an input graph $G = (V, E)$, a set $T \subseteq V$ of terminals and an integer k . We want to ask whether there exists a set $X \subseteq V \setminus T$ of size at most k such that any path between two different terminals intersects X .

We use the following lemma in Section 5.2.

Proposition 5.1.3 ([67]). *Let T be a tree and $x, y, z \in V(T)$. Then there exists a vertex $v \in V(T)$ such that every connected component of $T - v$ has at most one vertex from $\{x, y, z\}$.*

5.2 Semi Clique Tree Decomposition

Given a graph G and an integer k , we aim to construct a $(4, 7k + 5)$ -semi clique tree decomposition \mathcal{T} of G or conclude that G has no CVD of size at most k . We loosely

follow the ideas used for the tree decomposition algorithm in [147] to construct a tree decomposition of a graph G of width at most $4\text{tw}(G) + 4$, where $\text{tw}(G)$ is the treewidth of G . But before that, we propose the following lemmas that we use in getting the required $(4, 7k + 5)$ -semi clique tree decomposition.

Lemma 5.2.1. *Let G be a graph having a CVD of size k . Then G has a $(1, k)$ -semi clique tree decomposition.*

Proof. Let Y be the chordal vertex deletion set of G of size k . Since $G - Y$ is a chordal graph, it has a clique tree decomposition \mathcal{T}' . Adding Y to each bag of the tree decomposition \mathcal{T}' , we get a $(1, k)$ -semi clique tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ of G . \square

Lemma 5.2.2. *For a graph G on n vertices with a CVD of size k , the number of maximal cliques in G is bounded by $\mathcal{O}(2^k \cdot n)$. Furthermore, there is an algorithm that given any graph G either concludes that there is no CVD of size k in G or enumerates all the maximal cliques of G in $\mathcal{O}(2^k \cdot n^{\omega+1})$ time where ω is the matrix multiplication exponent.*

Proof. Let $X \subseteq V(G)$ be of size at most k such that $G - X$ is a chordal graph. For any maximal clique C in G let $C_X = C \cap X$ and $C_{G-X} = C \setminus X$. Since $G - X$ is a chordal graph, it has at most $n - k$ maximal cliques [80].

We claim that for a subset $C_X \subseteq X$ and a maximal clique Q in $G - X$, there is at most one subset $Q' \subseteq Q$ such that $C_X \cup Q'$ forms a maximal clique in G . If there are two distinct subsets Q_1, Q_2 of Q such that $C_X \cup Q_1$ and $C_X \cup Q_2$ are cliques in G , then $C_X \cup Q_1 \cup Q_2$ is a clique larger than the cliques $C_X \cup Q_1$ and $C_X \cup Q_2$. Thus, since there are at most 2^k subsets of X and at most n maximal cliques in G , the total number of maximal cliques in G is upper bounded by $2^k(n - k)$.

There is an algorithm that given a graph H , enumerates all the maximal cliques of H with $\mathcal{O}(|V(H)|^\omega)$ delay (the maximum time taken between outputting two consecutive solutions) [125]. If G has a CVD of size k , there are at most $2^k n$ maximal cliques in G

which can be enumerated in $\mathcal{O}(2^k n^{\omega+1})$ time. So the algorithm to enumerate clique runs for at most $2^k n + 1$ rounds, if we note that the number of maximal cliques enumerated is more than $2^k n$ then we return that G has no CVD of size k . \square

Lemma 5.2.3. *Let G be a graph having a CVD of size k and $w : V(G) \rightarrow \mathbb{R}_{\geq 0}$ be a weight function on $V(G)$. There exists a $\frac{2}{3}$ -balanced separation (A, B) of G with respect to w such that the graph induced on the corresponding separator $G[A \cap B]$ is a $(1, k)$ -semi clique.*

Proof. First we prove that there is a $\frac{1}{2}$ -balanced separator X such that $G[X]$ is a $(1, k)$ -semi clique. By Lemma 5.2.1, there is a $(1, k)$ -semi clique tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ of G . Arbitrarily root the tree of T at a node $r \in V(T)$. For any node $y \in V(T)$, let T_y denote the subtree of T rooted at node y and G_y denote the graph induced on the vertices of G present in the bags of nodes of T_y . That is $V(G_y) = \bigcup_{t \in V(T_y)} X_t$. Let t be the farthest node of T from the root r such that $w(V(G_t)) > \frac{1}{2}w(V(G))$. That is, for all nodes $t' \in V(T) \setminus \{t\}$, we have that $w(V(G_{t'})) \leq \frac{1}{2}w(V(G))$.

We claim that $X = X_t$ is a $\frac{1}{2}$ -balanced separator of G . Let t_1, \dots, t_p be the children of t . Since X is a bag of the tree decomposition \mathcal{T} , each of the connected components of $G - X$ are contained either in $G_{t_i} - X$ for some $i \in [p]$ or $G[V(G) \setminus V(G_t)]$. Since $w(V(G_t)) > \frac{1}{2}w(V(G))$, we have $w(V(G) \setminus V(G_t)) < \frac{1}{2}w(V(G))$. By the choice of t , we have $w(V(G_{t_i})) \leq \frac{1}{2}w(V(G))$ for all $i \in [p]$.

Now we define a $\frac{2}{3}$ -balanced separation (A, B) for G where the set $X = A \cap B$ is a $\frac{1}{2}$ balanced separator. Let D_1, \dots, D_q be the vertex sets of the connected components of $G - X$. Let $a_i = w(D_i)$ for all $i \in [q]$. Without loss of generality, assume that $a_1 \geq \dots \geq a_q$. Let q' be the smallest index such that $\sum_{i=1}^{q'} a_i \geq \frac{1}{3}w(V(G))$ or $q' = q$ if no such index exists. Clearly, $\sum_{i=q'+1}^q a_i \leq \frac{2}{3}w(V(G))$. We prove that $\sum_{i=1}^{q'} a_i \leq \frac{2}{3}w(V(G))$. If $q' = 1$, $\sum_{i=1}^{q'} a_i = a_{q'} \leq \frac{1}{2}w(V(G))$ and we are done. Else, since q' is the smallest index such that $\sum_{i=1}^{q'} a_i \geq \frac{1}{3}w(V(G))$, we have $\sum_{i=1}^{q'-1} a_i < \frac{1}{3}w(V(G))$. We also note that $a_{q'} \leq a_{q'-1} \leq \sum_{i=1}^{q'-1} a_i < \frac{1}{3}w(V(G))$. Hence $\sum_{i=1}^{q'} a_i = \sum_{i=1}^{q'-1} a_i + a_{q'} \leq \frac{2}{3}w(V(G))$.

Now we define $A = X \cup \bigcup_{i \in [q']} D_i$ and $B = X \cup \bigcup_{i \in [q] \setminus [q']} D_i$. Notice that $X = A \cap B$ and (A, B) is a separation of G . Also notice that $w(A \setminus B) = \sum_{i=1}^{q'} a_i \leq \frac{2}{3}w(V)$ and $w(B \setminus A) = \sum_{i=q'+1}^q a_i \leq w(V(G)) - \frac{1}{3}w(V(G)) = \frac{2}{3}w(V(G))$ as $\sum_{i=1}^{q'} a_i \geq \frac{1}{3}w(V(G))$. Since X is a bag of the tree decomposition \mathcal{T} , $G[X]$ is a $(1, k)$ -semi clique. \square

Using Lemmas 5.2.2 and 5.2.3, we obtain the following corollary.

Corollary 5.2.1. *Let G be a graph with a CVD of size k . Let $N \subseteq V(G)$ with $5k + 3 \leq |N| \leq 6k + 4$. Then there exists a partition (N_A, N_B) of N and a vertex subset $X \subseteq V(G)$ satisfying the following properties.*

- $|N_A|, |N_B| \leq 4k + 2$.
- X is a vertex separator of N_A and N_B in the graph G .
- $G[X]$ is a $(1, k)$ -semi clique.

Moreover, there is an algorithm that given any graph G , either concludes that there is no CVD of size k in G or computes such a partition (N_A, N_B) of N and the set X in $\mathcal{O}(2^{7k} \cdot (kn^3 + n^{\omega+1}))$ time.

Proof. Let us define a weight function $w : V(G) \rightarrow \mathbb{R}_{\geq 0}$ such that $w(v) = 1$ if $v \in N$ and 0 otherwise. From Lemma 5.2.3, we know that there exists a pair of vertex subsets (A, B) which is the balanced separation of G with respect to w where the graph induced on the corresponding separator $G[A \cap B]$ is a $(1, k)$ -semi clique.

Let us define the partition (N_A, N_B) . We add $(A \setminus B) \cap N$ to N_A and $(B \setminus A) \cap N$ to N_B . Since (A, B) is a balanced separation of G with respect to w , $|(A \setminus B) \cap N|, |(B \setminus A) \cap N| \leq \frac{2}{3}|N| \leq 4k + 2$. For each vertex $u \in (A \cap B) \cap N$, we iteratively add u to the currently smaller of the two sets of N_A and N_B . Since $|N| \leq 6k + 4 \leq 2 \cdot (4k + 2)$, we have $|N_A|, |N_B| \leq 4k + 2$ even after this process. This shows the existence of subsets N_A, N_B and $X = A \cap B$. But the proof

is not constructive as the existence of (A, B) uses the $(1, k)$ -semi clique tree decomposition of G which requires the chordal vertex deletion set.

We now explain how to compute these subsets without the knowledge of a $(1, k)$ -semi clique tree decomposition of G . Let $X = C'' \uplus N''$ where C'' is a clique and $|N''| \leq k$. We go over all $2^{|N|} \leq 2^{6k+4}$ 2-partitions of N to guess the partition (N_A, N_B) . We then use Lemma 5.2.2 to go over all maximal cliques D of G . Then we apply the classic Ford-Fulkerson maximum flow algorithm to find the separator Z of the sets N_A and N_B in the graph $G[V \setminus D]$. If $|Z| > k$, we can conclude that G has no CVD of size k in G . Otherwise, in one such iteration, it is the case that $C'' \subseteq D$ and $Z \subseteq N'' = X \setminus C''$.

Thus, we obtained a set $X' = D \uplus Z$ such that $G[X']$ is a $(1, k)$ -semi clique and X' is a vertex separator of N_A and N_B in the graph G .

Now we estimate the time taken to obtain these sets. We first go over all $\mathcal{O}(2^k \cdot n)$ maximal cliques of the graph which takes $\mathcal{O}(2^k \cdot n^{\omega+1})$ time. Then for each of the $\mathcal{O}(2^k \cdot n)$ maximal cliques, we go over at most 2^{6k+4} guesses for N_A and N_B . Finally, we use the Ford-Fulkerson maximum flow algorithm to find the separator of size at most k for N_A and N_B which takes $\mathcal{O}(k(n+m))$ time. Overall the running time is $\mathcal{O}(2^k \cdot n^{\omega+1} + (2^k n) \cdot 2^{6k} \cdot (k(n+m))) = \mathcal{O}(2^{7k} \cdot (kn^3 + n^{\omega+1}))$. \square

Lemma 5.2.4. *Let G be a graph having a CVD of size k . Let C_1, C_2, C_3 be three distinct cliques in G . Then there exists a vertex subset $X \subseteq V(G)$ such that $G[X]$ is a $(1, k)$ -semi clique and X is a separator of C_i and C_j for all $i, j \in \{1, 2, 3\}$ and $i \neq j$. Moreover, there is an algorithm that given any graph G , either concludes that there is no CVD of size k in G or computes X in $\mathcal{O}(4^k \cdot (kn^3 + n^{\omega+1}))$ time.*

Proof. By Lemma 5.2.1, there is a $(1, k)$ -semi clique tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ of G . By Proposition 5.1.1, we know that there exist nodes $t_1, t_2, t_3 \in V(T)$ such that $C_1 \subseteq X_{t_1}$, $C_2 \subseteq X_{t_2}$ and $C_3 \subseteq X_{t_3}$. If two of the three nodes t_1, t_2, t_3 is the same node t , then it can be easily seen that $X = X_t$ is the required separator as only at most one

of C_1, C_2 , and C_3 remains after its deletion.

Hence assume that all three nodes t_1, t_2, t_3 are distinct. From Proposition 5.1.3, we know that there exists a node $t \in V(T)$ such that (i) t_1, t_2 and t_3 are in different connected components of $T - t$. We claim that $X = X_t$ is the required separator. Since X is a bag in the $(1, k)$ -semi clique tree decomposition \mathcal{T} , $G[X]$ is a $(1, k)$ -semi clique. Because of statement (i), we have that X is a separator of C_i and C_j for all $i, j \in \{1, 2, 3\}$ and $i \neq j$. The proof is not constructive as we do not have a $(1, k)$ -semi clique tree decomposition of G .

We compute a set X' such that $G[X']$ is a $(1, k)$ -semi clique and X' is a separator of C_i and C_j for all $i, j \in \{1, 2, 3\}$ and $i \neq j$, without the knowledge of a $(1, k)$ -semi clique tree decomposition of G . Let $X = C'' \uplus N''$ where C'' is a clique and $|N''| \leq k$. Using Lemma 5.2.2, we either conclude that G has no CVD of size k or we go over all the maximal cliques of the graph G . We know that $C'' \subseteq D$ for one of such maximal cliques D . Now in the graph $G[V \setminus D]$, we know that there exists a set $Z \subseteq N'' = X \setminus C''$ of size at most k which separates the cliques $C_x \setminus D, C_y \setminus D$ and $C_z \setminus D$. To find Z , we add three new vertices x', y' and z' . We make x' adjacent to all the vertices of $C_x \setminus D$, y' adjacent to all the vertices of $C_y \setminus D$ and z' adjacent to all the vertices of $C_z \setminus D$. We find the node multiway cut Y of size at most k with the terminal set being $\{x', y', z'\}$. The set Y can be found in $\mathcal{O}(2^k km)$ using the known algorithm for node multiway cut [50, 89]. If the algorithm returns that there is no such set Y of size k , we conclude that there is no CVD of size at most k in G . Else we get a set $X' = D \uplus Y$ which satisfies the properties of X .

Now we estimate the time taken to obtain X' . We get all the $\mathcal{O}(2^k \cdot n)$ maximal cliques of the graph in $\mathcal{O}(2^k \cdot n^{\omega+1})$ time. Now for each maximal clique we use the $\mathcal{O}(2^k km)$ algorithm for node multiway cut. Thus, the overall running time is $\mathcal{O}(2^k \cdot n^{\omega+1} + (2^k n) \cdot (2^k km)) = \mathcal{O}(4^k \cdot (kn^3 + n^{\omega+1}))$. \square

Now we prove our main result (i.e., Theorem 21) in this section. For convenience, we

restate it here.

Theorem 1 (21). *There is an algorithm that given a graph G and an integer k runs in time $\mathcal{O}(2^{7k} \cdot (kn^4 + n^{\omega+2}))$ and either constructs a $(4, 7k + 5)$ -semi clique tree decomposition \mathcal{T} of G or concludes that there is no chordal vertex deletion set of size k in G . Moreover, the algorithm also provides a partition $C_1 \uplus C_2 \uplus C_3 \uplus C_4 \uplus N$ of each bag of \mathcal{T} such that $|N| \leq 7k + 5$ and C_i is a clique in G for all $i \in \{1, 2, 3, 4\}$.*

Proof. We assume that G is connected as if not we can construct a $(4, 7k + 5)$ -semi clique tree decomposition for each connected component of G and attach all of them to a root node whose bag is empty to get the required $(4, 7k + 5)$ -semi clique tree decomposition of G .

To construct a $(4, 7k + 5)$ -semi clique tree decomposition \mathcal{T} , we define a recursive procedure $\text{Decompose}(W, S, d)$ where $S \subset W \subseteq V(G)$ and $d \in \{0, 1, 2\}$. The procedure returns a rooted $(4, 7k + 5)$ -semi clique tree decomposition of $G[W]$ such that S is contained in the root bag of the tree decomposition. The procedure works under the assumption that the following invariants are satisfied.

- $G[S]$ is a $(d, 6k + 4)$ -semi clique and $W \setminus S \neq \emptyset$.
- $S = N_G(W \setminus S)$. Hence S is called the *boundary* of the graph $G[W]$.

To get the required $(4, 7k + 5)$ -semi clique tree decomposition of G , we call $\text{Decompose}(V(G), \emptyset, 0)$ which satisfies all the above invariants. The procedure $\text{Decompose}(W, S, d)$ calls procedures $\text{Decompose}(W', S', d')$ and a new procedure $\text{SplitCliques}(W', S')$ whenever $d = 2$. For these subprocedures, we will show that $|W' \setminus S'| < |W \setminus S|$. Hence by induction on the cardinality of $W \setminus S$, we will show the correctness of the Decompose procedure.

The procedure $\text{SplitCliques}(W, S)$ with $S \subset W \subseteq V(G)$ also outputs a rooted $(4, 7k + 5)$ -semi clique tree decomposition of $G[W]$ such that S is contained in the root bag of the tree

decomposition. But the invariants under which it works are slightly different which we list below.

- $G[S]$ is a $(3, 5k + 3)$ -semi clique and $W \setminus S \neq \emptyset$.
- $S = N_G(W \setminus S)$.

Notice that the only difference between invariants for Decompose and SplitCliques is the first invariant where we require $G[S]$ to be a $(3, 5k + 3)$ -semi clique for SplitCliques and $(d, 6k + 4)$ -semi clique for Decompose.

The procedure SplitCliques(W, S) calls procedures Decompose($W', S', 2$) where we will again show that $|W' \setminus S'| < |W \setminus S|$. Hence again by induction on cardinality of $W \setminus S$, we will show the correctness. Now we describe how the procedure Decompose is implemented.

Implementation of Decompose(W, S, d): Notice that $d \in \{0, 1, 2\}$. Firstly, if $|W \setminus S| \leq k + 1$, we output the tree decomposition as a node r with bag $X_r = W$ and stop. Clearly the graph $G[X_r]$ is a $(4, 7k + 5)$ -semi clique and it contains S . Otherwise, we do the following.

We construct a set \hat{S} with the following properties.

1. $S \subset \hat{S} \subseteq W \subseteq V(G)$.
2. $G[\hat{S}]$ is a $(d + 1, 7k + 5)$ -semi clique. Let $\hat{S} = C' \uplus N'$ where $G[C']$ is the union of $d + 1$ cliques and $|N'| \leq 7k + 5$.
3. Every connected component of $G[W \setminus \hat{S}]$ is adjacent to at most $5k + 3$ vertices of N' .

Since $G[S]$ is a $(d, 6k + 4)$ -semi clique, we have that $S = C \uplus N$, where $G[C]$ is the union of d cliques and $|N| \leq 6k + 4$.

Case 1: $|N| < 5k + 3$. We set $\hat{S} = S \cup \{u\}$, where u is an arbitrary vertex in $W \setminus S$. Note that this is possible as $W \setminus S \neq \emptyset$. Clearly \hat{S} follows all the properties above.

Case 2: $5k + 3 \leq |N| \leq 6k + 4$. Note that $G[W]$ being a subgraph of G also has a chordal vertex deletion set of size at most k if G has it. Applying Corollary 5.2.1 for the graph $G[W]$ and the subset N , we either conclude that G has no CVD of size k or get a partition (N_A, N_B) of N , a subset $X \subseteq W$ and a partition $D \uplus Z$ of X , where D is a clique in $G[W]$ and $|Z| \leq k$, in time $\mathcal{O}(2^{7k} \cdot (kn^3 + n^{\omega+1}))$ such that $|N_A|, |N_B| \leq 4k + 2$ and X is a vertex separator of N_A and N_B in the graph $G[W]$.

We define $\hat{S} = S \cup X \cup \{u\}$ where u is an arbitrary vertex in $W \setminus S$. We need to verify that \hat{S} satisfies the required properties.

Claim 5.2.1. *The set \hat{S} satisfies properties (1), (2) and (3).*

Proof. Since $u \in W \setminus S$, $S \subset \hat{S}$. Hence \hat{S} satisfies property (1).

We now show that \hat{S} satisfies property (2). Recall that $S = C \uplus N$, where $G[C]$ is the union of d cliques and $|N| \leq 6k + 4$. We define sets $C' = C \cup D$ and $N' = ((N \cup Z) \setminus C') \cup \{u\}$. Notice that $\hat{S} = C' \cup N'$. Clearly $G[C']$ is the union of $d + 1$ cliques. Also $|N'| \leq |N| + |Z| + 1 \leq (6k + 4) + k + 1 \leq 7k + 5$. Thus \hat{S} satisfies property (2).

We now show that \hat{S} satisfies property (3). Recall $\hat{S} = C' \cup N'$, where $C' = C \cup D$ and $N' = ((N \cup Z) \setminus C') \cup \{u\}$. Recall that $X = D \cup Z \subseteq \hat{S}$ is separator of N_A and N_B . where $N = N_A \uplus N_B$ and $|N_A|, |N_B| \leq 4k + 2$. This implies that any connected component H in $G[W \setminus X]$ can contain at most $4k + 2$ vertices from N as the neighborhood of $V(H)$ is contained in X , because X is a separator. Moreover $|Z| \leq k$. This implies that any connected component in $G[W \setminus \hat{S}]$ is adjacent to at most $4k + 2$ vertices in N and at most k vertices in Z , and hence at most $5k + 3$ vertices in $N' = ((N \cup Z) \setminus C') \cup \{u\}$. \square

Now we define the recursive subproblems arising in the procedure Decompose (W, S, d) using the constructed set \hat{S} . If $\hat{S} = W$, then there will not be any recursive subproblem. Otherwise, let P_1, P_2, \dots, P_q be vertex sets of the connected components of $G[W \setminus \hat{S}]$ and $q \geq 1$ because $\hat{S} \neq W$. We have the following cases:

Case 1: $d < 2$: For each $i \in [q]$, recursively call the procedure $\text{Decompose}(W' = N_G[P_i], S' = N_G(P_i), d + 1)$.

We now show that the invariants are satisfied for procedures $\text{Decompose}(W' = N_G[P_i], S' = N_G(P_i), d + 1)$ for all $i \in [q]$. We start by noticing that since $d < 2$, $d + 1 \leq 2$ which is required for the validity of the procedure. Let $Q_i = S' \cap N'$. Note that from condition (3) for \hat{S} , we have $|Q_i| \leq 5k + 3$. Since $S' \setminus Q_i \subseteq C'$ and $G[C']$ is a union of $d + 1$ cliques, $G[S']$ forms a $(d + 1, 5k + 3)$ -semi clique which is also a $(d + 1, 6k + 4)$ -semi clique. Also by definition of neighbourhoods, $P_i = N_G[P_i] \setminus N_G(P_i) = W' \setminus S'$. Since P_i is a non-empty set by definition, $W' \setminus S'$ is non-empty. Hence the first invariant required for the Decompose is satisfied. Since $S' = N_G(P_i) = N_G(N_G[P_i] \setminus N_G(P_i)) = N_G(W' \setminus S')$, the second invariant is satisfied.

Case 2: $d = 2$: For each $i \in [q]$, recursively call the procedure $\text{SplitCliques}(W' = N_G[P_i], S' = N_G(P_i))$. We can show that the invariants for SplitCliques are satisfied with the proofs similar to the previous case.

We now explain how to construct the $(4, 7k + 5)$ -semi clique tree decomposition using $\text{Decompose}(W, S, d)$. Here, we assume that $\text{Decompose}(W', S', d + 1)$ and $\text{SplitCliques}(W', S')$ return a $(4, 7k + 5)$ -semi clique tree decomposition $G[W']$ when $|W' \setminus S'| < |W \setminus S|$. That is, we apply induction on $|W \setminus S|$. Look at the subprocedures $\text{Decompose}(W', S', d)$ and $\text{SplitCliques}(W', S')$. We have $W' \setminus S' = N_G[P_i] \setminus N_G(P_i) = P_i$ which is a subset of $W \setminus \hat{S}$ which in turn is a strict subset of $W \setminus S$. Hence $|W' \setminus S'| < |W \setminus S|$. Hence we apply induction on $|W \setminus S|$ to the subprocedures. Let \mathcal{T}_i be the $(4, 7k + 5)$ -semi clique tree decomposition obtained from the subprocedure with $W' = N_G[P_i]$ and $S' = N_G(P_i)$. Let r_i be the root of \mathcal{T}_i whose associated bag is X_{r_i} . By induction hypothesis $S' \subseteq X_{r_i}$. We create a node r with the corresponding bag $X_r = \hat{S}$. For each $i \in [q]$, we attach \mathcal{T}_i to r by adding edge (r, r_i) . Let us call the tree decomposition obtained so with root r as \mathcal{T} . We return \mathcal{T} as the output of $\text{Decompose}(W, S, d)$. By construction, it easily follows that \mathcal{T} is a $(4, 7k + 5)$ -semi clique tree decomposition of the graph $G[W]$ with the

root bag containing S . We note that when $W = \hat{S}$, the procedure returns a single node tree decomposition with $X_r = W = \hat{S}$.

Implementation of SplitCliques Procedure: Again if $|W \setminus S| \leq k + 1$, we output the tree decomposition as a node r with bag $X_r = W$ and stop. Clearly the graph $G[X_r]$ is a $(4, 7k + 5)$ -semi clique and it contains S . Otherwise we do the following. Let $S = C \uplus N = (C_x \uplus C_y \uplus C_z) \uplus N$ where C_x, C_y and C_z are the vertex sets of the three cliques in $G[C]$. We apply Lemma 5.2.4 to graph $G[W]$ and sets C_x, C_y and C_z , to either conclude that G has no CVD of size k or obtain a set Y such that Y separates the sets C_x, C_y and C_z and $G[Y]$ is a $(1, k)$ -semi clique. Let $Y = D \uplus X$ where D is a clique and $|X| \leq k$.

Let $Y' = Y \cup \{u\}$ where u is any arbitrary vertex from $W \setminus S$ which we know to be non-empty. If $S \cup Y' = W$, then it will not call any recursive subproblem. Otherwise, let P_1, P_2, \dots, P_q be the connected components of the graph $G[W \setminus (S \cup Y')]$. We recursively call $\text{Decompose}(W' = N_G[P_i], S' = N_G(P_i), 2)$ for all $i \in [q]$.

Since Y' is a separator of the cliques C_x, C_y and C_z , any connected component P_i will have neighbours to at most one of the three cliques $C_x \setminus Y', C_y \setminus Y'$ and $C_z \setminus Y'$ in $G[W \setminus (S \cup Y')]$. We show that the invariants required for the procedure Decompose are satisfied in these subproblems. Let us focus on the procedure $\text{Decompose}(W' = N_G[P_i], S' = N_G(P_i), 2)$ which has neighbours only to the set $C_x \setminus Y'$. We define sets $C' = C_x \cup D$ and $N' = (N \cup X \cup \{u\}) \setminus C'$. The vertex set P_i has neighbours only to the set $(C_x \uplus N) \cup Y' = (C_x \uplus N) \cup (D \uplus X) \cup \{u\} = (C_x \cup D) \cup (N \cup X \cup \{u\}) = C' \uplus N'$. Clearly $G[C']$ is the union of at most two cliques and $|N'| \leq |N| + |X| + 1 = 5k + 3 + k + 1 \leq 6k + 4$. Hence the first invariant is satisfied for the procedure $\text{Decompose}(N_G[P_i], N_G(P_i), 2)$. The proof of the second invariant is the same as that of the subproblems of Decompose procedure. The satisfiability of invariants for other subprocedures can also be proven similarly.

We now construct the $(4, 7k + 5)$ -semi clique tree decomposition returned by $\text{SplitCliques}(W, S)$. Again we apply induction on $|W \setminus S|$. Consider the subprocedures $\text{Decompose}(W', S', d)$. We have $W' \setminus S' = N_G[P_i] \setminus N_G(P_i) = P_i$ which is a subset of

$W \setminus (S \cup Y')$ which in turn is a strict subset of $W \setminus S$ as $u \in W \setminus S$ is present in Y' . Hence $|W' \setminus S'| < |W \setminus S|$ and we apply induction on $|W \setminus S|$ to the subprocedures. Let \mathcal{T}_i be the $(4, 7k + 5)$ -semi clique tree decomposition obtained from the subprocedure with $W' = N_G[P_i]$ and $S' = N_G(P_i)$. Let r_i be the root of \mathcal{T}_i whose bag X_{r_i} we show contains S' . We create a node r with the corresponding bag $X_r = S \cup Y' = (C_x \uplus C_y \uplus C_z \uplus D) \uplus N'$. For each $i \in [q]$, we attach \mathcal{T}_i to r by adding edge (r, r_i) . Let us call the tree decomposition obtained so with root r as \mathcal{T} . We return \mathcal{T} as the output of $\text{SplitCliques}(W, S, d)$. By construction, it easily follows that \mathcal{T} is a $(4, 7k + 5)$ -semi clique tree decomposition of the graph $G[W]$ with the root bag containing S . We mention that when $W = S \cup Y'$, the procedure returns a single node tree decomposition with $X_r = W$.

Running time analysis: In the procedure Decompose , we invoke Corollary 5.2.1 which takes $\mathcal{O}(2^{7k} \cdot (kn^3 + n^{\omega+1}))$ time. For the procedure SplitCliques , we invoke Lemma 5.2.4 which takes $\mathcal{O}(4^k \cdot (kn^3 + n^{\omega+1}))$ time. All that is left is to bound the number of calls of the procedures Decompose and SplitCliques . Each time Decompose or SplitCliques is called, it creates a set \hat{S} (in the case of SplitCliques , $\hat{S} = S \cup Y'$) which is a strict superset of S . This allows us to map each call of Decompose or SplitCliques to a unique vertex $u \in \hat{S} \setminus S$ of $V(G)$. Hence the total number of calls of Decompose and SplitCliques is not more than the total number of vertices n . Hence the overall running time of the algorithm which constructs the $(4, 7k + 5)$ -semi clique tree decomposition of G is $\mathcal{O}(2^{7k} \cdot (kn^4 + n^{\omega+2}))$. \square

Faster Algorithm. We can get a faster algorithm by making use of the fact that any C_4 -free graphs have $\mathcal{O}(n^2)$ maximal cliques[61]. The algorithm first repeatedly finds induced subgraphs of G which have C_4 if present and removes all its vertices from G . Let Z denote the union of the vertices removed in this process. We now apply the algorithm in Theorem 21 on the graph $G - Z$. Note that the graph $G - Z$ has $\mathcal{O}(n^2)$ maximal cliques. This drops the $2^k n$ factor to n^2 in the running times of the algorithms of Corollary 5.2.1 and 5.2.4 invoked in the algorithm of Theorem 21. Hence the running time to obtain the semi clique tree decomposition \mathcal{T}' of $G - Z$ drops to $\mathcal{O}(2^{6k} \cdot (kn^5 + n^{\omega+2}))$. We now obtain a semi

clique tree decomposition \mathcal{T} of G from \mathcal{T}' by adding Z to every bag of \mathcal{T}' .

We now prove that \mathcal{T} is a $(4, 7k + 5)$ -semi clique tree decomposition. Let $k_1 \leq k$ denote the number of disjoint C_4 's present in Z . Hence $|Z| = 4k_1$. Since Z has k_1 disjoint C_4 's, any CVD of G contains at least k_1 vertices of Z . Hence G has a CVD of size k if and only if $G - Z$ has a CVD of size $k - k_1$. The algorithm of Theorem 21 either concludes that $G - Z$ has no CVD of size $k - k_1$ or returns a $(4, 7(k - k_1) + 5)$ -semi clique tree decomposition \mathcal{T}' of $G - Z$. In the former case we can correctly conclude that G has no CVD of size k . In the later case, we now obtain a semi clique tree decomposition \mathcal{T} of G from \mathcal{T}' by adding Z to every bag of \mathcal{T}' . The vertices which are not part of the cliques in every bag of \mathcal{T} is $7(k - k_1) + 5 + |Z| = 7(k - k_1) + 5 + 4k_1$ which is at most $7k + 5$. Hence \mathcal{T} is a $(4, 7k + 5)$ -semi clique tree decomposition. Hence we have the following theorem.

Theorem 22. *There is an algorithm that given a graph G and an integer k runs in time $\mathcal{O}(2^{6k} \cdot (kn^5 + n^{\omega+2}))$ and either constructs a $(4, 7k + 5)$ -semi clique tree decomposition \mathcal{T} of G or concludes that there is no chordal vertex deletion set of size k in G . Moreover, the algorithm also provides a partition $C_1 \uplus C_2 \uplus C_3 \uplus C_4 \uplus N$ of each bag of \mathcal{T} such that $|N| \leq 7k + 5$ and C_i is a clique in G for all $i \in \{1, 2, 3, 4\}$.*

We note that though the above algorithm is faster, it does not improve the running time of algorithms of the dynamic programming algorithms in Section 4. This is because these algorithms store solutions for every possible subset of the non-clique part of the bags which is at least 2^{7k} . Hence in the following section, we continue using the algorithm in Theorem 21.

5.3 Structural Parameterizations with Chordal Vertex Deletion Set

Theorem 23. *d -COLORABLE SUBGRAPH BY CVD can be solved in*

$d^{4d+7k+5}2^{3(7k+5)}n^{\mathcal{O}(d)}$ time.

Proof. First, we use Theorem 21 to construct a $(4, 7k + 5)$ -semi clique tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ of G in $\mathcal{O}^*(2^{7k})$ time. Now we use the dynamic programming algorithm on tree decompositions given by Fomin and Golovach (Theorem 1 of [65]) on \mathcal{T} to find the maximum sized induced subgraph H of G such that H is d -colorable. Note that the set $V(G) \setminus V(H)$ is the solution that we are looking for and if its size is at most ℓ , we return YES. Else we return NO.

This dynamic programming algorithm defines a state $cost(t, S, c)$ for all nodes $t \in V(T)$, subsets $S \subseteq X_t$ such that $G[S]$ is d -colorable and a function $c : S \rightarrow [d]$. Since each bag X_t of \mathcal{T} is a $(4, 7k + 5)$ semi clique, at most d vertices of each clique can be part of S as else there is a presence of a $(d + 1)$ sized clique in S which is not d -colorable. Hence we can bound the size of S as $4d + 7k + 5$ and also bound the number of possible subsets S as $n^{4d}2^{7k+5}$. Number of possible functions c is at most $d^{|S|}$ which is at most $d^{4d+7k+5}$. Hence we bound the number of states as $d^{4d+7k+5}2^{7k+5}n^{\mathcal{O}(d)}$. For each state, the time taken is $\mathcal{O}(|S|^2)$. Hence the overall running time is $d^{4d+7k+5}2^{3(7k+5)}n^{\mathcal{O}(d)}$. \square

Corollary 5.3.1. VERTEX COVER BY CVD and ODD CYCLE TRANSVERSAL BY CVD can be solved in $2^{21k}n^{\mathcal{O}(1)}$ and $2^{28k}n^{\mathcal{O}(1)}$ time, respectively.

We can directly use the dynamic programming on bounded treewidth to get algorithms with better running times for VERTEX COVER BY CVD and ODD CYCLE TRANSVERSAL BY CVD and for FEEDBACK VERTEX SET BY CVD using the fact that any vertex cover contains all but one from each clique and any odd cycle transversal and feedback vertex set contains all but two from each clique.

Theorem 24. Given a graph G and an integer k , there exist algorithms that determine that G has no CVD of size k or

- find a minimum vertex cover in $2^{7k}n^{\mathcal{O}(1)}$ time, and

- find a minimum odd cycle transversal in $3^{7k}n^{\mathcal{O}(1)}$ time, and
- find a minimum feedback vertex set in $2^{\omega 7k}n^{\mathcal{O}(1)}$ time.

Proof. First, we use Theorem 21 to construct a $(4, 7k + 5)$ -semi clique tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ of G in $\mathcal{O}^*(2^{7k})$ time. Arbitrarily root the tree T at a node r . Let $X_t = C_{t,1} \uplus \dots \uplus C_{t,4} \uplus N_t$ where $|N_t| \leq 7k + 5$ and $C_{t,j}$ is a clique in G for all $j \in \{1, \dots, 4\}$. In the tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$, for any vertex $t \in V(T)$, we call D_t to be the set of vertices that are descendant of t . We define G_t to be the subgraph of G on the vertex set $X_t \cup \bigcup_{t' \in D_t} X_{t'}$.

Proof sketch of the algorithm for VERTEX COVER BY CVD:

We briefly explain the dynamic programming (DP) table entries on \mathcal{T} . In a standard DP for each node $t \in V(T)$ and $Y \subseteq X_t$, we have a table entry $DP[Y, t]$ which stores the size of a minimum vertex cover S of G_t such that $Y = X_t \cap S$ and if no such vertex cover exists, then $DP[Y, t]$ stores ∞ . We only need to store $DP[Y, t]$ whenever it is not equal to ∞ . Now consider a bag X_t in \mathcal{T} . For any $Y \subseteq X_t$, if $|C_{t,j} \setminus Y| \geq 2$ for any $j \in [4]$, then $DP[Y, t] = \infty$ because $C_{t,j}$ is a clique. Therefore, we only need to consider subsets $Y \subseteq X_t$ for which $|C_{t,j} \setminus Y| \leq 1$ for all $j \in [4]$. The number of choices of such subsets Y is bounded by $\mathcal{O}(2^{7k}n^4)$. This implies that the total number of DP table entries is $\mathcal{O}(2^{7k}n^5)$. All these values can be computed in time $\mathcal{O}(2^{7k}n^{\mathcal{O}(1)})$ time using standard dynamic programming in a bottom up fashion. For more details about dynamic programming over tree decomposition, see [45].

Proof sketch of the algorithm for ODD CYCLE TRANSVERSAL BY CVD:

Any odd cycle transversal contains all but at most two vertices from each clique $C_{1,j}$, $i \in [4]$. Using this fact we can bound the number of DP table entries to be at most $3^{7k}n^{\mathcal{O}(1)}$. Then, by computing the entries in a bottom up fashion in time $3^{7k}n^{\mathcal{O}(1)}$ using standard

arguments.

Proof sketch of algorithm for FEEDBACK VERTEX SET BY CVD:

We use the ideas from the DP algorithm for FEEDBACK VERTEX SET using the rank-based approach [19]. We give a more detailed algorithm as the techniques for solving FEEDBACK VERTEX SET parameterized by treewidth are slightly sophisticated and it may not be obvious for the reader that these techniques extend to semi-clique tree decompositions.

We create an auxiliary graph G' by adding a vertex v_0 to G and making it adjacent to all the vertices of G . Let E_0 be the set of newly added edges. Thus we add v_0 to all the bags to get the tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ of G' . We use a dynamic programming algorithm for FEEDBACK VERTEX SET on \mathcal{T} where the number of entries of the DP table we will show to be $2^{7k+5}n^{11}$. Let $X_t = C_{t,1} \uplus \dots \uplus C_{t,4} \uplus N_t$ for all $t \in V(T)$ where $|N_t| \leq 7k+5$ and $C_{t,j}$ is a clique in G for all $j \in \{1, \dots, 4\}$. For a node $t \in V(T)$, a subset $Y \subseteq X_t$ and integers $i, j \in [n]$, we define the entry $DP[t, Y, i, j]$. The entry $DP[t, Y, i, j]$ stores a partition \mathcal{P} of Y if

- there exists a vertex subset $X \subseteq D_t$, $v_0 \in X$ such that $X \cap X_t = Y$ and
- there exists an edge subset $X_0 \subseteq E(G_t) \cap E_0$ such that in the graph $(X, E(G_t[X \setminus \{v_0\}]) \cup X_0)$, we have i vertices, j edges, no connected component is fully contained in $D_t \setminus X_t$ and the elements of Y are connected according to the partition \mathcal{P} .

We set $DP[t, Y, i, j] = \infty$ if the entry can be inferred to be invalid from Y .

We claim that the FEEDBACK VERTEX SET BY CVD instance (G, k) is a yes instance if and only if for the root r of \mathcal{T} with $X_r = \{v_0\}$ and some $i \geq |V| - \ell$, we have $DP[r, \{v_0\}, i, i-1]$ to be non-empty. In the forward direction, we have a feedback vertex set W of size ℓ . The graph $G - W$ has $|V| - \ell$ vertices and $|V| - \ell - c$ edges where c is the number of connected components of $G - W$. We define $X = V \setminus W \cup \{v_0\}$ and X_0 to be c edges connecting v_0 to any one of the vertices of each of the c components of $V \setminus W$. We have

$|X| \geq |V| - \ell$. The graph $(X, E(G_t[X \setminus \{v_0\}]) \cup X_0)$ has $|V| - \ell$ edges and satisfies the properties required for an entry in $DP[r, \{v_0\}, i, i-1]$. In the reverse direction, we have a graph $(X, E(G_t[X \setminus \{v_0\}]) \cup X_0)$ having i edges and $i-1$ edges. Since no connected component of the graph can be contained in $V(G_t) \setminus \{v_0\}$, the graph is a tree. Hence $V \setminus X$ is a feedback vertex set.

Now we give the recurrence relations for computing $DP[t, Y, i, j]$. For this purpose, we convert \mathcal{T} into a nice tree decomposition which can be done in $\mathcal{O}(n^3)$ time. Since each bag contains cliques of size $\mathcal{O}(n)$, the number of nodes of \mathcal{T} can also blow up to be $\mathcal{O}(n^3)$ with $\mathcal{O}(n^2)$ new nodes possibly added for each and edge of \mathcal{T} corresponding to $\mathcal{O}(n^2)$ edges and vertices added or removed to obtain the collection of the cliques in the child node from the collection in the parent node.

The recurrences for computing $DP[t, Y, i, j]$ more or less remains the same as in [19]. Before we state them, we define some operations on a family \mathcal{A} of partitions of a universe U .

- Union: For two families \mathcal{A} and \mathcal{B} of partitions of U , we define the union $\mathcal{A} \cup \mathcal{B}$ as the family obtained by taking the union of both families.
- Insert: For a family of partitions \mathcal{A} and set X such that $X \cap U = \emptyset$, $insert(X, \mathcal{A})$ is the family of partitions obtained by adding each element of X as singleton sets in each of the partitions of \mathcal{A} .
- Glue: For elements u, v , $glue(uv, \mathcal{A})$ is obtained by combining the sets containing u and v in each of the partitions of \mathcal{A} .
- Project: For a family of partitions \mathcal{A} and set $X \subseteq U$, $project(X, \mathcal{A})$ is the family obtained by removing all the elements of X from each of the partitions, but discarding the partition if doing so reduces the number of sets in the partition.
- Join: For a partition P of universe U and Q of universe U' , the join of P and Q is defined as follows. Look at a graph G over vertices $U \cup U'$. Look at each set S in P

and turn the corresponding vertex set into a clique. We do so for all the sets in P as well as Q . Now, look at the set of connected components of G . We get a partition of $U \cup U'$ with each set of the partition being the vertex set of the corresponding connected component. This partition is called the join of P and Q .

For a family of partitions \mathcal{A} over U and a family of partitions \mathcal{B} over U' , $join(\mathcal{A}, \mathcal{B})$ is the family of partitions over $U \cup U'$ obtained by taking the join of each pair of partitions from \mathcal{A} and \mathcal{B} .

We have the following recurrence relations.

- Leaf Node: We set the entry $DP[t, \phi, 0, 0] = \{\phi\}$ and all other entries as invalid.
- Introduce vertex Node: Let t and t' be the parent and child nodes with vertex v being introduced in t . We have

$$DP[t, Y, i, j] = \begin{cases} \infty & \text{if } v = v_0 \text{ and } v \notin Y \\ insert(v, DP[t', Y \setminus \{v\}, i-1, j]) & \text{if } v \in Y \\ DP[t', Y, i, j] & \text{otherwise} \end{cases}$$

The first case is to ensure that if the vertex introduced is v_0 , then it has to be in Y . Otherwise, if $v \in Y$, we extend solutions that do not contain v with $i-1$ vertices by adding singleton v to each of the partitions.

- Forget vertex node: Let t and t' be the parent and child nodes with vertex v being forgotten in t . We have

$$DP[t, Y, i, j] = DP[t', Y, i, j] \cup project(v, DP[t', Y \cup \{v\}, i, j])$$

We extend solutions of child node t' for both the cases when v is present or absent in the corresponding set Y . We do so by taking the union of partitions for both cases. In

the case when v is present, we make sure that the partitions where v is a singleton are not added. This is because the corresponding component can no longer be connected as it has no intersection with X_t .

- Introduce Edge Node: Let t and t' be the parent and child nodes with edge uv being added in t . We have

$$DP[t, Y, i, j] = \begin{cases} DP[t', Y, i, j] \cup glue(v_0v, DP[t', Y, i, j-1]) & \text{if } u = v_0 \text{ and } v \in Y \\ DP[t', Y, i, j] \cup glue(v_0u, DP[t', Y, i, j-1]) & \text{if } v = v_0 \text{ and } u \in Y \\ glue(uv, DP[t', Y, i, j-1]) & \text{if } u, v \in Y \\ DP[t', Y, i, j] & \text{otherwise} \end{cases}$$

If $u = v_0$ and $v \in Y$ (or the symmetric case), then the edge v_0v may or may not be part of the maximal induced forest corresponding to the solution. Hence we can choose to insert v_0v or not. Else if both u and v are present in Y , then edge uv has to be present in the maximal induced forest. In the cases where the edge is present, we obtain the corresponding family of partitions by gluing sets containing u and v for each partition.

- Join node: Let t be the parent node and t', t'' be the child nodes. We have

$$DP[t, Y, i, j] = \bigcup_{i_1+i_2=i-|Y|, j_1+j_2=j} join(DP[t', Y, i_1, j_1], DP[t'', Y, i_2, j_2])$$

A pair of vertices x, y in X_t is connected in G_t if there is a vertex $z \in X_t$ such that x and z is connected in $G_{t'}$ and y and z is connected in $G_{t''}$. The partition of Y corresponding to this connectivity is obtained exactly via the join of pair of partitions in the entries of t' and t'' .

Now we bound the number of table entries and the number of partitions stored for each

entry. Consider a bag X_t in \mathcal{T} . For any $Z \subseteq X_t$, We focus on the sets $C_{t,j} \setminus Z$ which is part of the forest. If $|C_{t,j} \setminus Z| \geq 3$ for any $j \in [4]$, then $DP[t, Y, i, j] = \infty$ because $G[C_{t,j}]$ contains a triangle as $C_{t,j}$ is a clique. Therefore, we only need to consider subsets $Z \subseteq X_t$ for which $|C_{t,j} \setminus Z| \leq 2$ for all $j \in [4]$. Hence we have $|C_i| \leq 2$. The number of choices for each C_i is at most $|C_{t,i}| \leq n^2$. We also have $|Y| \leq |N_t| \leq 7k + 5$. Since the number of nodes of \mathcal{T} is $\mathcal{O}(n^3)$, we have the total number of DP table entries is $\mathcal{O}(2^{7k} n^{13})$. In each DP table entry $DP[t, Y, i, j]$, we store partitions of Y . The cardinality of Y is bounded by $7k + 13$ as $|C_{t,j} \setminus Y| \leq 2$ for all $j \in [4]$. Hence the number of partitions stored in a particular entry $DP[t, Y, i, j]$ can be as huge as $|Y|^{\mathcal{O}(|Y|)}$ which is bounded by $(7k + 13)^{\mathcal{O}(k)}$. But as we will see below, we devise a reducing routine that allows us to only store at most 2^{7k+13} partitions in each table entry.

We use the ideas from [19] to bound the time taken to compute all the table entries of a particular node t . In particular, for each table entry $DP[t, Y, i, j]$, after obtaining a family \mathcal{A} of partitions over a universe U using recurrence relations, we use reducing algorithm Theorem 3.7 of [19] to obtain a subfamily \mathcal{A}' of size $2^{|U|}$ which “represents” \mathcal{A} . Since the subfamily \mathcal{A}' represents \mathcal{A} , it can be used for further evaluations in the dynamic programming algorithm. For more details, we refer to [19].

Let us first bound the time taken to compute the recurrence relations. Using Proposition 3.3 of [19], given to families \mathcal{A} and \mathcal{B} over a universe U , the time taken for every operation is bounded by $|\mathcal{A}||\mathcal{B}||U|^{\mathcal{O}(1)}$. The leading factor is the time taken for the reducing algorithm Theorem 3.7 of [19] which is bounded by $\mathcal{O}(|\mathcal{A}| \cdot 2^{(\omega-1)|U|} |U|^{\mathcal{O}(1)})$ where ω is the matrix multiplication exponent. Since the number of table entries is $\mathcal{O}(2^{7k} n^{13})$, U is at most $7k + 13$ and \mathcal{A} is at most $2^{|U|}$, we have the total time bounded to be $\mathcal{O}(2^{(\omega-1)7k} (7k)^{\mathcal{O}(1)} 2^{7k} n^{13}) = \mathcal{O}(2^{\omega 7k} (7k)^{\mathcal{O}(1)} n^{13})$. \square

5.3.1 SETH Lower Bounds

A graph G is called a cluster graph if it is a disjoint union of complete graphs. It can be seen that all cluster graphs are chordal. We define a problem called VERTEX COVER BY CLSVD.

VERTEX COVER BY CLSVD

Input: A graph $G = (V, E)$, $k, \ell \in \mathbb{N}$ and a set $S \subseteq V(G)$ with $|S| \leq k$ such that $G[V \setminus S]$ is a cluster graph.

Parameter: k

Question: Is there a vertex cover of size ℓ in G ?

Assuming SETH, we show that VERTEX COVER BY CLSVD, FVS BY CVD and OCT BY CVD cannot have an $\mathcal{O}^*((2 - \varepsilon)^k)$ FPT algorithm. As the class of all cluster graphs is a subclass of the class of chordal graphs, deletion distance to a chordal graph is a smaller parameter. Hence the lower bound also holds for VERTEX COVER BY CVD.

To show the following theorem, we give a parameterized reduction from HITTING SET parameterized by the size of the universe n to VERTEX COVER BY CLSVD and use the fact that assuming SETH, HITTING SET cannot be solved in $\mathcal{O}^*((2 - \varepsilon)^n)$ time.

Theorem 25. VERTEX COVER BY CLSVD cannot be solved in $\mathcal{O}^*((2 - \varepsilon)^k)$ time for any $\varepsilon > 0$ assuming SETH.

Proof. We give a reduction from HITTING SET defined as follows.

HITTING SET : In any instance of HITTING SET, we are given a set of elements U with $|U| = n$, a family of subsets $\mathcal{F} = \{F \subseteq U\}$ and a natural number k . The objective is to find a set $F \subseteq U$, $|F| \leq k$ such that $S \cap F \neq \emptyset$ for all $S \in \mathcal{F}$.

The problem cannot be solved in $\mathcal{O}^*((2 - \varepsilon)^n)$ time assuming SETH [48].

Consider a HITTING SET instance (U, \mathcal{F}) . We construct an instance of VERTEX COVER

BY CLSVD as follows. For each element $u \in U$, we add a vertex v_u . For each set $S \in \mathcal{F}$, we add $|S|$ vertices corresponding to the elements in S . We also make the vertices of S into a clique. Finally, for each element $u \in U$, we add edges from v_u to the vertex corresponding to u for each set in \mathcal{F} that contains u . See Figure 5.1.

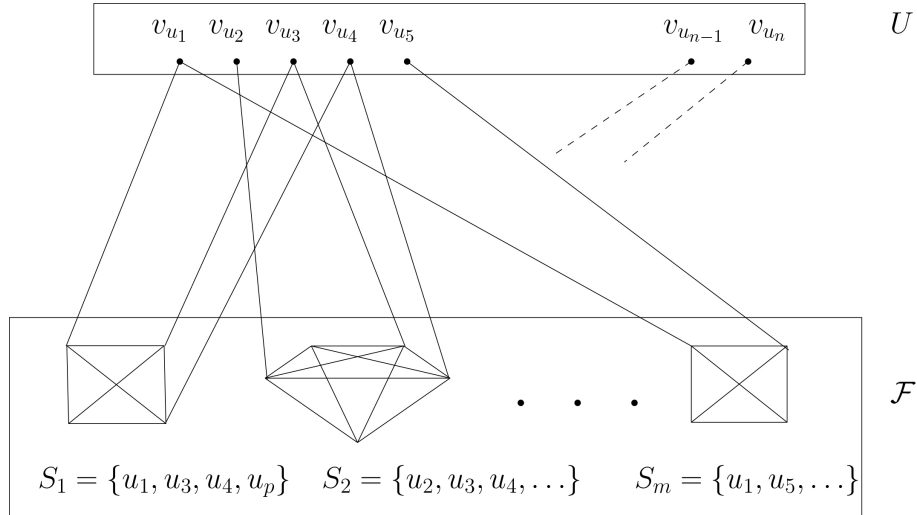


Figure 5.1: Reduction from HITTING SET to VERTEX COVER BY CLSVD

Note that the set of vertices $\bigcup_{u \in U} v_u$ forms a cluster vertex deletion set of size n for the graph G we constructed.

We claim that there is a hitting set of size k in the instance (U, \mathcal{F}) if and only if there is a vertex cover of size $k + \sum_{S \in \mathcal{F}} (|S| - 1)$ in G .

Let $X \subseteq U$ be the hitting set of size k . For each set $S \in \mathcal{F}$, mark an element of X which intersects S . Now we create a subset of vertices Y in G consisting of vertices corresponding to elements in X plus the vertices corresponding to all the unmarked elements in S for every set $S \in \mathcal{F}$. Clearly $|Y| = k + \sum_{S \in \mathcal{F}} (|S| - 1)$. We claim that Y is a vertex cover of G . Let us look at an edge of G between an element vertex u and its corresponding copy vertex in S containing u . If u is unmarked in S , then it is covered as the vertex corresponding to u in S is present in Y . If it is marked, then the element v_u is present in Y which covers the edge. All the other edges of G have both endpoints in a set $S \in \mathcal{F}$. Since one of them is unmarked, it belongs to Y which covers the edge.

Conversely, let Z be a vertex cover of G of size $k + \sum_{S \in \mathcal{F}} (|S| - 1)$. Since the graph induced on vertices of set S forms a clique for each $S \in \mathcal{F}$, Z should contain all the vertices of the clique except one to cover all the edges of the clique. Let us mark these vertices. This means that at least $\sum_{S \in \mathcal{F}} (|S| - 1)$ of the vertices of Z are not element vertices v_u . Now the remaining k vertices of Z should hit all the remaining edges in G . Suppose it contains another vertex x corresponding to an element u in set $S \in \mathcal{F}$. Since x can only cover the edge from x to the element vertex v_u out of the remaining edges, we could remove x and add v_u as it is not present in Z and still get a vertex cover of G of the same size. Hence we can assume, without loss of generality that all the remaining vertices of Z are element vertices v_u . Let X' be the union of the k elements corresponding to these element vertices. We claim that X' is a hitting set of (U, \mathcal{F}) of size k . Suppose X' does not hit a set $S \in \mathcal{F}$. Look at the unmarked vertex x in the vertices of S . There is an edge from x to its element vertex v_u . Since $u \notin X'$, this edge is uncovered in G giving a contradiction.

Hence given a HITTING SET instance (U, \mathcal{F}) , we can construct an instance for VERTEX COVER BY CLSVD with parameter n . Hence, if we could solve VERTEX COVER BY CLSVD in $\mathcal{O}^*((2 - \varepsilon)^k)$ time, we can solve HITTING SET in $\mathcal{O}^*((2 - \varepsilon)^n)$ time contradicting SETH. \square

The proof of the following theorem works by modifying the reduction in the above proof to replace edges with triangles.

Theorem 26. *FVS BY CVD and OCT BY CVD given the modulator cannot be solved in $\mathcal{O}^*((2 - \varepsilon)^k)$ time for any $\varepsilon > 0$ assuming SETH.*

Proof. To prove the above theorem, we again give a reduction very similar to the reduction given in the proof of Theorem 25. Consider a HITTING SET instance (U, \mathcal{F}) . To create an instance of FEEDBACK VERTEX SET BY CVD or ODD CYCLE TRANSVERSAL BY CVD, we replace each edge in the above reduction by a triangle. It can be easily shown that the graph obtained after removing the vertices corresponding to elements in U forms a chordal

graph. The proof follows on similar lines. □

5.4 Conclusion

Our main contribution is to develop techniques for addressing structural parameterization problems when the modulator is not given. The question, of Fellows et al. about whether there is an FPT algorithm for VERTEX COVER parameterized by perfect deletion set with only a promise on the size of the deletion set, is open. Regarding problems parameterized by chordal deletion set size, though our algorithms are based on treewidth DP, we remark that not all problems that have FPT algorithms when parameterized by treewidth necessarily admit an FPT algorithm parameterized by CVD. For example, DOMINATING SET parameterized by treewidth admits an FPT algorithm [45] while DOMINATING SET parameterized by CVD is para-NP-hard as the problem is NP-hard in chordal graphs [22]. Generalizing our algorithms for other problems, for example, for the optimization problems considered by Liedloff et al. [111] would be an interesting direction.

Finally, we believe that this whole notion of permissive problems needs to be explored in many facets of structural parameterizations where finding the modulator is more expensive than solving the problem when the modulator is given.

Chapter 6

Fixed-parameter tractability of $(n - k)$

List Coloring

6.1 Introduction

The graph coloring problem is one of the fundamental combinatorial optimization problems with applications in scheduling, register allocation, pattern matching and many other active research areas. Given a graph $G = (V, E)$, the k -coloring problem is asking whether there is a way to assign at most k colors/labels to vertices of a graph such that no two adjacent vertices share the same color. Such a coloring is also known as a *proper k -coloring*. The smallest number of colors needed to color a graph G is called its chromatic number, and is denoted by $\chi(G)$. Determining whether a graph is 3-colorable is NP-hard [78] while the 2-coloring problem has a linear time algorithm. It is even hard to approximate the chromatic number in polynomial time. The 3-coloring problem remains NP-complete even on 4-regular planar graphs[52]. There are some generalizations and variations of ordinary graph colorings which are motivated by practical applications such as PRECOLORING EXTENSION and LIST COLORING. In this chapter, we focus on the LIST COLORING problem defined as follows.

LIST COLORING PROBLEM

Input: A graph $G = (V, E)$ and a LIST L of $|V|$ sets of colors with $L(v)$ being the entry for $v \in V$

Question: Is there an assignment of colors $c : V \rightarrow \cup_{v \in V} L(v)$ such that it respects the list L , i.e. for any vertex v , $c(v) \in L(v)$ and for any two adjacent vertices u and v , $c(v) \neq c(u)$?

A list L is ℓ -REGULAR if each set contains exactly ℓ colors. ℓ -REGULAR LIST COLORING problem is to decide whether $G = (V, E)$ has a coloring that respects L , where L is ℓ -REGULAR.

Note that when all the lists $L(v) = \{1, 2, \dots, k\}$, the problem becomes the k -COLORING problem.

Literature and Previous Work. As 3-coloring is NP-hard, the k -coloring problem is para-NP-hard when parameterized by the number of colors. Hence various other parameterizations have been studied for the COLORING problem. Some include structural parameterizations like the size of the vertex cover [95], treewidth [45], deletion distance to a graph class \mathcal{G} where COLORING is solvable in polynomial time such as bipartite graphs, chordal graphs, complete graphs[33, 138].

It is interesting to see if the COLORING problem is FPT for some parameterization, whether LIST COLORING problem which is a generalization of COLORING also has an FPT algorithm. For example, q -COLORING problem parameterized by the vertex cover size k has a $q^k n^{\mathcal{O}(1)}$ algorithm. The same result can be extended to q -REGULAR LIST COLORING [95]. But there are also parameterizations where an FPT result in COLORING problem does not extend to LIST COLORING. For example, while COLORING is FPT when parameterized by the treewidth of the graph [45], LIST COLORING problem is $W[1]$ -hard for the same parameter [63]. See [138] for a summary of results on parameterizations of COLORING and LIST COLORING.

It has been long known that it is fixed-parameter tractable to determine whether a graph can be colored with at most $n - k$ colors (here k is the parameter); i.e. whether one can save k colors from the trivial n coloring of the graph [38]. We ask whether this result can be generalized to LIST COLORING by asking whether $(n-k)$ -REGULAR LIST COLORING is FPT parameterized by k . A previous result by Arora and a subset of authors [10] showed that $(n-k)$ -REGULAR LIST COLORING is in XP. In this chapter, we improve this result by showing that the problem is in FPT. A crucial part of the work in this chapter is to show that the problem is FPT when the graph is $f(k)$ vertices away from a clique that fits in the deletion distance parameterization techniques.

6.2 Preliminaries

Definition 6.2.1. (*Crown Decomposition*) A crown decomposition of a graph G is a partitioning of $V(G)$ into sets C, H and R such that

- C is non-empty.
- C induces an independent set in G .
- There are no edges from C to R .
- G contains a matching of size $|H|$ between C and H .

Here C is said to be the crown and H is the head. (See Figure 6.1).

Theorem 27. (*Hall's theorem, [55]*) Let G be an undirected bipartite graph with bipartition V_1 and V_2 . The graph G has a matching saturating V_1 if and only if for all $X \subseteq V_1$, we have $|N(X)| \geq |X|$.

Theorem 28. [86] Let $G = (V, E)$ be an undirected bipartite graph with bipartitions V_1 and V_2 . Then in $\mathcal{O}(|E|\sqrt{|V|})$ time we can either find a matching saturating V_1 or an inclusion wise minimal set $X \subseteq V_1$ such that $|N(X)| < |X|$.

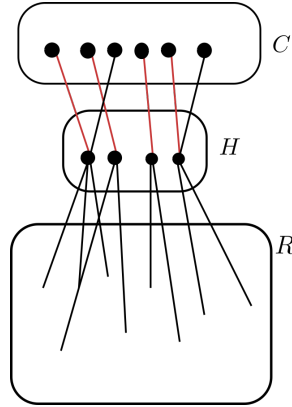


Figure 6.1: Crown Decomposition

Theorem 29. [16] LIST COLORING can be solved in $2^n n^{\mathcal{O}(1)}$ time.

6.3 FPT algorithm for $(n-k)$ -REGULAR LIST COLORING

We restate the following reduction rules and results from [10] without proofs.

Reduction Rule 4. [10] Delete any vertex with degree less than $(n - k)$.

Lemma 6.3.1. [10] If there exists a set of k colors using which it is possible to color at least $2k$ vertices of G respecting the lists in L , then there is a feasible coloring for G respecting L .

Lemma 6.3.2. [10] LIST COLORING PROBLEM is polynomial time solvable on a clique.

We keep applying Reduction Rule 4 till it is no longer applicable and hence from now onwards we assume that every vertex has degree at least $(n - k)$. We can also assume that $n \geq 3k$ for otherwise, we can apply Theorem 29 to obtain a fixed-parameter tractable algorithm with running time $2^{3k} n^{\mathcal{O}(1)}$.

Let $C = \cup_{v \in V} L(v)$. We create a bipartite graph $G_B(V, C, E)$ with bipartization (V, C) . There is an edge between v and a color c if $c \in L(v)$.

We start with the following new reduction rule.

Reduction Rule 5. Let C' be an inclusion wise minimal subset of C such that $|N(C')| < |C'|$ in the graph G_B . Delete all the vertices in $N(C')$ from G .

Lemma 6.3.3. Reduction Rule 5 is safe and can be implemented in polynomial time.

Proof. Let $D = C' \setminus \{c\}$ for any arbitrary vertex $c \in C'$. Since C' is an inclusion wise minimal set satisfying the condition of the rule, for any subset $D' \subseteq D$, $|N(D')| \geq |D'|$. Hence by Hall's theorem 27, there is a matching saturating D .

Let M be a matching saturating D into $N(D)$. As $N(D) \subseteq N(C')$, we have $|N(D)| \leq |N(C')|$. Since $|N(D)| \geq |D| = |C'| - 1$, we have $|N(C')| \geq |C'| - 1$. But as $|N(C')| \leq |C'| - 1$ by definition of C' , we get $|N(C')| = |C'| - 1$. By definition of D , we have $|N(D)| = |D|$ and $N(D) = N(C')$. Hence there are no unmatched vertices in $N(C')$ with respect to M . We have a crown decomposition in the graph G_B with D as the crown and $N(D)$ as the head.

Let us denote for each vertex $v \in N(C')$, m_v as the matching partner in D with respect to M . We show that there exists a list coloring respecting L in G if and only if there exists a list coloring respecting L in $G \setminus N(C')$. Since $G \setminus N(C')$ is a subgraph of G , the forward direction is true. In the converse, suppose that there exists a coloring \mathcal{C} respecting L in the graph $G \setminus N(C')$. We extend this coloring \mathcal{C} to a coloring \mathcal{C}' in G by assigning color m_v to each vertex $v \in N(C')$. We claim that \mathcal{C}' is a proper coloring respecting L . Suppose not. Then there exists a monochromatic edge $(u, v) \in E$. Since \mathcal{C} is a valid coloring, either $u \in N(C')$ or $v \in N(C')$. Since all the vertices in $N(C')$ have different colors, both u and v cannot be in $N(C')$. Hence without loss of generality, assume $v \notin N(C')$. Since all the vertices of $N(C')$ are colored by using colors in C' and $v \notin N(C')$, color of v cannot be $m_u \in C'$ giving a contradiction. \square

Note that by deleting the vertices in $N(C')$ from G , we are also deleting the colors C' from C as the colors in C' are only present in the set $N(C')$ by definition.

We use Theorem 28 in the graph G_B to either conclude that there is a matching saturating

C or obtain a set C' such that $|N(C')| < |C'|$ in polynomial time. If its the latter, we apply Reduction Rule 5.

Note that when $|V| < |C|$, there is no matching in the bipartite graph G_B saturating C . Hence by Theorem 28, there exists a non-empty set C' which is inclusion wise minimal subset of C such that $|N(C')| < |C'|$. Hence Reduction Rule 5 can be applied reducing $|V|$ and $|C|$. When the rule can no longer be applied $|V| \geq |C|$. But note that if it is the case that $|V| = |C|$ when the reduction rule can no longer be applied, the matching M saturating C from Theorem 28 gives a perfect matching M in G_B . In this case, we can conclude that the input is a YES instance as we can construct a feasible list coloring function \mathcal{C} respecting L with $\mathcal{C}(v) = m_v$ where $v \in V$ and m_v the matching partner of v in M .

Hence we can assume henceforth that in the graph G , $|V| = n > |C|$.

For an edge $e = (u, v) \in \overline{G}$ we define a list $L(e) = L(u) \cap L(v)$. We call a matching M in \overline{G} a MULTICOLOR MATCHING if it is possible to choose a distinct color from each $L(e)$ for every edge $e \in M$. Now, we have the following corollary of Lemma 6.3.1 as the $2k$ end points of the k matching edges can be colored with k colors.

Corollary 29.1. *If there exists a multicolor matching of size k in \overline{G} , then there is a feasible coloring for G with respecting L .*

Next, we show that we can color each vertex of G with a different color, or there exists a MULTICOLOR MATCHING of size k or there exists a large clique in G .

Lemma 6.3.4. *For any $u, v \in V$, $|N_{G_B}(u) \cap N_{G_B}(v)| > n - 2k$ in G_B .*

Proof. Let $u, v \in V$. We have

$$\begin{aligned}
n = |V| &> |C| \\
&\geq |N_{G_B}(u) \cup N_{G_B}(v)| \\
&= |N_{G_B}(u)| + |N_{G_B}(v)| - |N_{G_B}(u) \cap N_{G_B}(v)| \\
&\geq 2n - 2k - |N_{G_B}(u) \cap N_{G_B}(v)|
\end{aligned}$$

from which it follows that $n > 2n - 2k - |N_{G_B}(u) \cap N_{G_B}(v)|$ from which the claim follows. \square

Next, we prove the following.

Lemma 6.3.5. *Either there is a multicolor matching of size k in \overline{G} or there is a clique of size $n - 2k$ in G .*

Proof. Find any maximal matching M in \overline{G} . Suppose $|M| < k$. Let V_M be the set of endpoints of edges in M . The set of vertices $V \setminus V_M$ is an independent set in \overline{G} , therefore they form a clique in G . Since $|V \setminus V_M| > n - 2k$, one part of the lemma follows.

If $|M| \geq k$, choose exactly k edges of the matching and let us call this set of edges as M . From Lemma 6.3.4 we know that between any pair of vertices in V , there are at least $n - 2k$ shared colors. As we have assumed that $n \geq 3k$, we have $n - 2k \geq k$. Hence we can greedily assign an unassigned color to each edge of M starting from an arbitrary color for the first edge in M , resulting in a multicolor matching of size k in \overline{G} . \square

If there is a multicolor matching of size k in \overline{G} , then by Corollary 29.1, G can be list colored. Now, in what follows, we show that the list coloring can be determined in FPT time when G has a clique of size at least $n - 2k$. Towards this, we define the graph class $\text{CLIQUE} + f(k)$ whose members G has the property that there exists a subset of $f(k)$ vertices in G

whose deletion results in a clique.

We look at $(n-k)$ -REGULAR LIST COLORING in the graph class $\text{CLIQUE} + f(k)$ parameterized by k . A recent result by Gutin et al. [81] gives a randomized $2^{f(k)}n^{O(1)}$ algorithm for (even a more general version of) this problem. However in the following result, we give a deterministic FPT algorithm using the fact that the degree of every vertex in our graph is at least $n - k$.

Notice that we have $f(k) = 2k$ in the case that we ended up with.

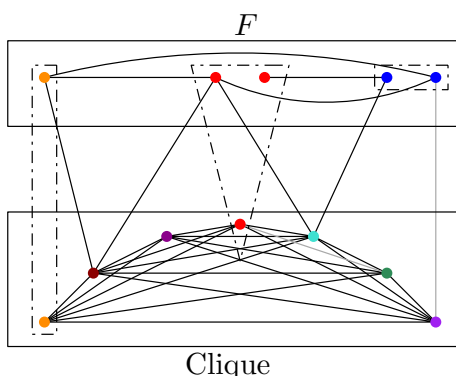


Figure 6.2: List coloring in $\text{clique} + f(k)$

Theorem 30. $(n-k)$ -REGULAR LIST COLORING is FPT for the graph class $\text{CLIQUE} + f(k)$.

Proof. Let $G(V, E)$ be a graph in $\text{CLIQUE} + f(k)$ such that $V = D \cup F$ where D induces a clique, and $|F| \leq f(k)$. Any feasible coloring for G partitions V into different color classes where each color class induces an independent set. Now we show that given any partition $\mathcal{V} = \{V_1, V_2 \dots V_l\}$ of V , in polynomial time, we can determine whether there is a list coloring of the vertices such that all vertices in $V_i \in \mathcal{V}$ are colored with a single color.

First, observe that the following properties must be satisfied for \mathcal{V} to be a partition of V into independent sets. We call a partition of V satisfying the following properties a *good partition* of V .

- Each subset V_i must be an independent set in G .

- For each subset $V_i \in \mathcal{V}$, $\bigcap_{v \in V_i} L(v) \neq \emptyset$.
- For each subset $V_i \in \mathcal{V}$, $|D \cap V_i| \leq 1$.

For a good partition \mathcal{V} , we create the following bipartite graph G_B with bipartization (\mathcal{V}, C) , where one partition \mathcal{V} contains a vertex corresponding to each V_i . Recall that $C = \bigcup_{v \in V} L(v)$ is the set of colors. There is an edge between the vertex corresponding to V_i and c_j if and only if $c_j \in \bigcap_{v \in V_i} L(v)$. Then the following claim is easy to see.

Claim 6.3.1. *There is a feasible coloring of G with exactly the color classes in a good partition \mathcal{V} if and only if there is a matching saturating \mathcal{V} in G_B .*

Proof. Let \mathcal{C} be a feasible coloring of G . For each color class $V_i \in \mathcal{V}$, we have a different color $\mathcal{C}(V_i)$. Hence by the definition of the edges of G_B , we have a matching saturating \mathcal{V} , the matching edges being $(V_i, \mathcal{C}(V_i))$.

In the converse, let M be a matching saturating \mathcal{V} . Let c_{V_i} be the matching partner of V_i in M . We construct a coloring function $\mathcal{C} : V \rightarrow C$ such that $\mathcal{C}(v) = c_{V_i}$ if $v \in V_i$. Since \mathcal{V} is a good partition, each $V_i \in \mathcal{V}$ is an independent set. Hence we have a feasible coloring of G . □

Now, we argue that the number of distinct good partitions of V is a function $g(k)$ to complete the argument.

Let $X = F \cup \overline{N}_{G \setminus F}(F)$ where $\overline{N}_{G \setminus F}(F) = \bigcup_{v \in F} \overline{N}_{G \setminus F}(v)$, i.e. the union of non neighbors in $V \setminus F$ for each vertex $v \in F$. Define $Y = V \setminus X$. Then every vertex of Y is adjacent to all other vertices of V . Hence each vertex of Y should get a separate color that is different from the colors of all other vertices of X in any proper coloring. Hence in a good partition \mathcal{V} , there will be a color class with the singleton element $\{y\}$ for every $y \in Y$. Let \mathcal{Y} denote the partition of Y of such singleton sets.

Hence to check if there exists a list coloring of G , it suffices to test for every partition \mathcal{X}

of X , whether the partition \mathcal{V} formed by the union of \mathcal{X} and \mathcal{Y} forms a feasible coloring. This can be tested using Claim 6.3.1 by going over all partitions of X .

The running time is bounded by $B_{|X|}n^{\mathcal{O}(1)}$ where $B_{|X|}$ is the number of partitions of X which is at most d^d where $|X| = d$. For any vertex v of F , $|\overline{N}_G(v)| \leq k$ due to reduction rule 4. Thus $|X| \leq f(k) + k \cdot f(k)$. Hence the overall runtime is $(f(k) + k \cdot f(k))^{(f(k)+k \cdot f(k))}n^{\mathcal{O}(1)}$.

□

From Lemma 6.3.5, we know that either there is a multicolor matching of size k in \overline{G} when we have a list coloring respecting the lists or that there is a clique of size at least $n - 2k$ in G where we can check if there is a list coloring in $(2k^2 + 2k)^{2k^2+2k}n^{\mathcal{O}(1)}$ time.

The running time can be slightly improved (to $(2k^2)^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$) by observing the following. Since the (non-neighbor) vertices in $\overline{N}(F)$ are part of a clique, in any valid coloring they need separate colors. Let us look at the subfamily of a partition of X containing at least one vertex from F . There are at most $2k$ sets in this subfamily and each set in the family contains at most one vertex from $\overline{N}(F)$. Hence we can fix a partition of X that corresponds to a list coloring by first fixing a partition of F and then adding at most one vertex from $\overline{N}(F)$ to each of the sets in this partition. The number of partitions of F is at most $2k^{2k}$. For a fixed partition of F , the number of ways in which we can add at most one vertex from $\overline{N}(F)$ to each of the sets is at most $(2k^2 + 1)^{2k}$. Hence the number of partitions of X is at most $(2k^2 + 1)^{2k} \cdot 2k^{2k}$. Thus we have the main result of the chapter.

Theorem 31. *$(n-k)$ -REGULAR LIST COLORING is FPT parameterized by k with running time $(2k^2)^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$.*

We give the following pseudo code below summarizing the entire algorithm.

Algorithm 6.3.1: FPT Algorithm for $(n-k)$ -REGULAR LIST COLORING

- 1 Input: A graph $G = (V, E)$ and a list L of $|V|$ many sets of $|V| - k$ colors with $L(v)$ being the entry for $v \in V$.
 - 2 Output: YES if there is a proper list coloring of V , NO otherwise.
 1. If $|V| \leq 3k$, use algorithm in Theorem 29.
 2. Repeat until it is no longer applicable :
 - Delete any vertex with degree less than $|V| - k$.
 - If there is a matching saturating C , return YES. Otherwise, find C' which is an inclusion wise minimal subset of C such that $|N(C')| < |C'|$ in the graph G_B . Delete all the vertices in $N(C')$ from G .
 3. Check if there is a multicolor matching of size k in \overline{G} . If so, return YES.
 4. Find a subset F of size at most $2k$ in G whose deletion results in a clique D .
 5. Compute $X = F \cup \overline{N}_{G \setminus F}(F)$.
 6. Go over all good partitions of X and see if there is a proper list coloring with the partition of X and singleton sets of $D \setminus X$ forming the color classes of the list coloring (using Theorem 30).
-

6.4 Conclusion

We have shown that $(n-k)$ -REGULAR LIST COLORING is FPT parameterized by k . Another well-studied notion in parameterized complexity is the notion of kernelization, where given an input instance (I, k) of the parameterized problem \mathcal{Q} , we use a polynomial time algorithm to convert it to an equivalent instance $(I', k') \in \mathcal{Q}$ where $|I'| \leq g(k)$ for some computable function g . The latter instance is called the kernel of the problem. A natural open problem is the existence of a polynomial kernel for $(n-k)$ -REGULAR LIST COLORING. Recently, Gutin et al. [81] solved this by giving a kernel with $\mathcal{O}(k^2)$ vertices and colors.

Chapter 7

Deletion Distance Parameterizations of Dominating Set Variants

7.1 Introduction

7.1.1 Motivation

DOMINATING SET problem is one of the classical NP-complete graph-theoretic problems. It asks for a minimum set of vertices in a graph such that every vertex is either in that set or has a neighbor in that set. It, along with several variations including *independent domination*, *total domination*, *efficient domination*, *connected domination*, *total perfect domination*, *threshold domination* are well-studied in all algorithmic paradigms including parameterized complexity and approximation and structural points of view. All of these versions are hard for the parameterized complexity class $W[2]$ in general graphs when parameterized by solution size. It means that the problem of determining whether a graph has a dominating set (or any variants listed above) of size k is unlikely to be fixed-parameter tractable.

We consider parameterizations of DOMINATING SET variants that are more natural and functions of the input graph. To the best of our knowledge, this is the first serious study of structural parameterization of any version of dominating set.

Our parameter of interest is the ‘distance’ of the graph from a natural class of graphs. Note that if dominating set is NP-hard in a graph class, then it will continue to be NP-hard even on graphs that are k away from the class, even for constant k (in particular for $k = 0$) and hence is unlikely to be fixed-parameter tractable. Hence it is natural to consider graphs that are not far from a class of graphs where the dominating set problem is polynomial time solvable. Our case study considers two such special graphs: cluster graphs where each connected component is a clique and split graphs where the vertex set can be partitioned into a clique and an independent set. In the former, all the variants of dominating set we consider are polynomial time solvable, while in the latter class of split graphs, we consider the independent and efficient dominating set problems that are polynomial time solvable. We call the set of vertices whose deletion results in a cluster graph and split graph as *cluster vertex deletion set* (CVD) and *split vertex deletion set* (SVD) respectively.

Finally, we remark that the size of minimum CVD and minimum SVD are at most the size of a minimum vertex cover in a graph, which is a well-studied parameterization in the parameter-ecology program [134].

It is called an *efficient dominating set* if for every vertex $v \in V$, $|N[v] \cap S| = 1$. Note that an efficient dominating set may not exist for a graph (for example, for a 4-cycle). If for every vertex v , $|N(v) \cap S| \geq r$, S is a *threshold dominating set* with threshold r . When $r = 1$, S is a *total dominating set*. Note that for dominating set, the vertices in S do not need other vertices to dominate them, but they do in a total dominating set. For more on these dominating set variants, see [83]. We will often denote *efficient dominating set* by EDS and *independent dominating set* by IDS in the rest of the article. When we say that a graph G is k -away from a graph in a graph class, what we mean is that there is a subset S of k vertices in the graph such that $G \setminus S$ belongs to the class.

Now we describe the main results in this chapter (See Table 7.1 for a summary). When parameterized by the deletion distance k to cluster graphs,

- we can find a minimum dominating set in $\mathcal{O}^*(3^k)$ time. Within the same time, we can also find a minimum independent dominating set (IDS) or a minimum efficient dominating set (EDS) or a minimum total dominating set. We also give a $\mathcal{O}^*((r+2)^k)$ algorithm for minimum threshold dominating set with threshold r . These algorithms are obtained through a dynamic programming approach for interesting generalizations of set cover which may be of independent interest. These results are discussed in Section 7.2.1.
- We complement our upper bound results by showing that for dominating set and total dominating set, $\mathcal{O}^*((2-\varepsilon)^k)$ algorithm is not possible for any $\varepsilon > 0$ under what is known as Set Cover Conjecture. We also show that for IDS, $\mathcal{O}^*((2-\varepsilon)^k)$ algorithm is not possible for any $\varepsilon > 0$ under the Strong Exponential Time Hypothesis (SETH) and for EDS no $2^{o(k)}$ algorithm is possible unless the Exponential Time Hypothesis (ETH) is false. It also follows from our reductions that dominating set, total dominating set and IDS do not have polynomial sized kernels unless $\text{NP} \subseteq \text{coNP}/\text{poly}$. These results are discussed in Section 7.2.2.

The standard dominating set and most of its variants are NP-hard or W[2]-hard in split graphs [145]. For the two variants IDS and EDS that are polynomial time solvable in split graphs, we show that when parameterized by the deletion distance k to split graphs,

- IDS can be solved in $\mathcal{O}^*(2^k)$ time and provide an $\mathcal{O}^*((2-\varepsilon)^k)$ lower bound for any $\varepsilon > 0$ under the strong exponential time hypothesis (SETH). We also show that IDS-SVD has no polynomial kernel unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.
- The 2^k barrier can be broken for EDS by designing an $\mathcal{O}^*(3^{k/2})$ algorithm. This is one of the very few problems with a runtime better than $\mathcal{O}^*(2^k)$ in the realm

	Cluster Deletion Set		Split Deletion Set	
	Algorithms	Lower Bounds	Algorithms	Lower Bounds
DS, TDS	$\mathcal{O}^*(3^k) \star$	$\mathcal{O}^*((2 - \varepsilon)^k)$ and $npk \star$		para-NP-hard
IDS	$\mathcal{O}^*(3^k) \star$	$\mathcal{O}^*((2 - \varepsilon)^k)$ and npk	$\mathcal{O}(2^k) \star$	$\mathcal{O}^*((2 - \varepsilon)^k)$ and npk
EDS	$\mathcal{O}^*(3^k) \star$	$\mathcal{O}^*(2^{o(k)}) \star$	$\mathcal{O}^*(3^{k/2}) \star$	$\mathcal{O}^*(2^{o(k)}) \star$
THDS	$\mathcal{O}^*((r + 2)^k) \star$	$npk \star$		para-NP-hard

Table 7.1: Summary of results. Results marked \star indicate our results. npk stands for ‘No polynomial kernel’

of structural parameterization. We also show that no $2^{o(k)}$ algorithm is possible unless the exponential time hypothesis (ETH) is false. These results are discussed in Section 7.3.

7.1.2 Related Work

Clique-width [43] of a graph is a parameter that measures how close to a clique the graph is. Courcelle et. al. [42] showed that for a graph with clique-width at most k , any problem expressible in MSO_1 (monadic second order logic of the first kind) has an FPT algorithm with k as the parameter if a k -expression for the graph (a certificate showing that the clique-width of the graph is at most k) is also given as input. The clique-width of a graph that is k away from a cluster graph can be easily shown to be $k + 1$ (with a k -expression) and all the dominating set variants discussed in this chapter can be expressed in MSO_1 and hence can be solved in FPT time in such graphs. But the running time function $f(k)$ in Courcelle’s theorem is huge (more than doubly exponential). Oum et al. [136] gave an $k^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ algorithm to solve the minimum dominating set for clique-width k graphs without assuming that the k -expression is given. There is a $\mathcal{O}^*(4^k)$ algorithm by Bodlaender et. al. [85] for finding minimum dominating set in graphs with clique-width k when the k -expression given as input. It is easy to construct the k -expression for graphs k away from a cluster graph and hence we have a $\mathcal{O}^*(4^k)$ algorithm. The algorithms we give

in Section 7.2, not only improve the running time, but also are applicable for other variants of dominating set.

7.1.3 Problem Definitions

SET COVER

Input: A universe \mathcal{U} and a family $\mathcal{F} \subseteq 2^{\mathcal{U}}$ and an integer k

Parameter: $|\mathcal{U}|$

Question: Does there exist k sets $A_1, \dots, A_k \in \mathcal{F}$ such that $\bigcup_{i=1}^k A_i = \mathcal{U}$?

CNF-SAT

Input: A boolean formula ϕ in conjunctive normal form with n variables and m clauses

Parameter: n

Question: Is there an assignment which evaluates ϕ to true?

3-CNF-SAT

Input: A boolean formula ϕ in conjunctive normal form with n variables and m clauses such that every clause has at most three literals.

Parameter: n

Question: Is there an assignment that evaluates ϕ to true?

We give a general template of formal definition of problems as follows:

\mathcal{P} - \mathcal{Q}

Input: An undirected graph $G = (V, E), S \subseteq V(G)$ which is a \mathcal{Q} and an integer ℓ .

Parameter: $|S|$

Question: Is there a \mathcal{P} in G of size at most ℓ ?

\mathcal{P} can be one of DOMINATING SET, EFFICIENT DOMINATING SET, INDEPENDENT DOMINATING SET, TOTAL DOMINATING SET and THRESHOLD DOMINATING SET denoted by DS, EDS, IDS, TDS and THDS. \mathcal{Q} can be one of CLUSTER VERTEX DELETION set, SPLIT VERTEX DELETION set and VERTEX COVER denoted by CVD, SVD and VC respectively.

For example, DS-CVD is as follows according to the template.

DS-CVD

Input: An undirected graph $G = (V, E), S \subseteq V(G)$ which is a cluster vertex deletion set and an integer ℓ .

Parameter: $|S|$

Question: Is there a dominating set in G of size at most ℓ ?

A minimal vertex cover is a vertex cover of a graph that is not a proper subset of any other vertex cover. We now define the following parameterizations of the maximum minimal vertex cover problem.

MMVC-VC

Input: An undirected graph $G = (V, E), S \subseteq V(G)$ such that S is a vertex cover and an integer ℓ .

Parameter: $|S|$

Question: Does G have a minimal vertex cover with at least ℓ vertices?

MMVC-CVD

Input: An undirected graph $G = (V, E), S \subseteq V(G)$ such that S is a cluster vertex deletion set of G and an integer ℓ .

Parameter: $|S|$

Question: Does G have a minimal vertex cover with at least ℓ vertices?

7.2 Dominating Set Variants parameterized by CVD Size

7.2.1 Upper Bounds

In a clique graph, any vertex of the graph is a dominating set. Hence in cluster graphs where each component is a clique, any optimal dominating set is such that it has exactly one vertex from each clique component. It is easy to see from the definitions that this dominating set is also efficient and independent. Any optimal threshold dominating set with threshold r is such that it contains exactly $r + 1$ vertices from each clique component. Note that in this case, every vertex has r neighbors excluding itself.

We can assume that the CVD set S of size k is given with the input. If not, we can use the algorithm by Boral et al. [23] that runs in $\mathcal{O}^*(1.92^k)$ time and either outputs a CVD set of size at most k or says that no such set exists.

We now give an algorithm for DS-CVD defined in Section 7.1.3.

Our FPT algorithm starts with making a guess S' for the solution's intersection with S . We delete vertices in $N[S'] \cap S$ as they have been already dominated by S' .

Let us denote the cliques in the cluster graph $G' = G \setminus S$ as C_1, C_2, \dots, C_q where $q \leq n - k$. We label the vertices of G' as $v_1, v_2, \dots, v_{|V \setminus S|}$ such that the first l_1 of them belong to the clique C_1 , the next l_2 of them belong to clique C_2 and so on for integers l_1, l_2, \dots, l_q . Note that for some cliques, it could be that all the vertices of the clique are dominated by S' . We are left with the problem of picking the minimum number of vertices from the clique vertices in $V \setminus S$ to dominate, the vertices of the cliques that are not yet dominated by S' , and the set $S \setminus N[S']$. We abstract out the problem as follows.

DS-DISJOINTCLUSTER

Input: An undirected graph $G = (V, E)$, $S \subseteq V$ such that every connected component of $G \setminus S$ is a clique, a $(0, 1)$ vector (f_1, f_2, \dots, f_q) corresponding to the cliques (C_1, \dots, C_q) and an integer ℓ .

Parameter: $|S|$

Question: Does there exist a subset $T \subseteq V \setminus S$ of size ℓ , that dominates all vertices of S and all vertices of all cliques C_i with flags $f_i = 1$?

For the DS-CVD problem, the set S in this new formulation is the remaining vertices of S after deleting $N[S'] \cap S$. Also for the clique C_i , the flag f_i is set to 1 if all its vertices have not been dominated by S' and is set to 0 otherwise. Note that if all the vertices of a clique component are not dominated, one of its vertices has to go into the solution to dominate the remaining vertices.

We now give an algorithm for DS-DISJOINTCLUSTER. We formulate this problem instance as a variant of SET COVER instance. We define the universe \mathcal{U} for the SET COVER instance as the vertex set S . For each vertex $v \in V \setminus S$, let $S_v = N(v) \cap S$. We define the family of sets of SET COVER as $\mathcal{F} = \{S_v | v \in V \setminus S\}$. We say that a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ covers a subset $W \subseteq \mathcal{U}$ if for every element $w \in W$, there exist some set in \mathcal{F}' containing w . A SET COVER solution $\mathcal{F}' \subseteq \mathcal{F}$ for $(\mathcal{U}, \mathcal{F})$ covers all the elements of \mathcal{U} . In the graph G , the vertices corresponding to the sets in \mathcal{F}' will dominate all the vertices in S . But DS-DISJOINTCLUSTER has the additional requirement of dominating the vertices of every clique C_i with $f_i = 1$ as well. This means from every such clique at least one vertex has to be in the solution. With this in mind, we define for each clique C_i in the graph G , a subfamily $\mathcal{B}_i = \{S_v : v \in C_i\}$ for the SET COVER instance. We call these subfamilies as *blocks*. Let us order the sets S_v in a block in the order of the vertices $v_1, \dots, v_{|V \setminus S|}$. We have the following problem which is a slight generalization of SET COVER.

SET-COVER WITH PARTITION

Input: A universe \mathcal{U} , a family of sets $\mathcal{F} = \{S_1, \dots, S_m\}$, a partition $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_q)$ of \mathcal{F} , a $(0, 1)$ vector (f_1, f_2, \dots, f_q) corresponding to each block in the partition $(\mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_q)$ and an integer ℓ .

Parameter: $|\mathcal{U}| = k$

Question: Does there exist a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ of size ℓ that covers \mathcal{U} and at least one set is present in blocks \mathcal{B}_i with flags $f_i = 1$?

Lemma 7.2.1. SET-COVER WITH PARTITION *can be solved in $\mathcal{O}^*(2^{|\mathcal{U}|})$ time.*

Proof. We give a dynamic programming algorithm to solve SET-COVER WITH PARTITION. The algorithm is similar to the exact algorithm to solve SET COVER [68] but with some modifications to handle the blocks \mathcal{B}_i with flag $f_i = 1$. For every subset $W \subseteq U$, for every $j \in [m]$ and a flag $f \in \{0, 1\}$, we define $OPT[W, j, f]$ as the cardinality of the minimum sized subfamily X of $\{S_1, \dots, S_j\}$ that covers W and from each block \mathcal{B}_i with $f_i = 1$, there is at least one set in X except the block \mathcal{B}_x containing the last set S_j where we reset the flag to f to indicate that at least f sets are required in that block.

We have $OPT[W, 1, f] = 1$ if $W \subseteq S_1$, else we set $OPT[W, 1, f]$ to ∞ to indicate that there is no such subfamily. To compute all the values of $OPT[W, j, f]$, we initially set all the remaining values to ∞ . Assuming that $OPT[W', j', f']$ is computed for all subsets $W' \subseteq W$, integers $1 \leq j \leq j$ and flags $f' \in \{0, 1\}$, we give the following recursive formulation for $OPT[W, j+1, f]$.

- Case 1 : S_{j+1} is not the first set in its block \mathcal{B}_x .

$$OPT[W, j+1, f] = \min \left\{ OPT[W, j, f], 1 + OPT[W \setminus S_{j+1}, j, 0] \right\}$$

In computing $OPT[W, j+1, f]$ recursively, either S_{j+1} is in the solution where the block \mathcal{B}_x is covered, and we are left to cover $W \setminus S_{j+1}$ using $\{S_1, \dots, S_j\}$ or S_{j+1}

is not picked and we are left to cover W using $\{S_1, \dots, S_j\}$. Hence we take the minimum of these two cases in the above recursive formulation.

- Case 2 : S_{j+1} is the first set in its block B_x .

$$OPT[W, j+1, f] = \begin{cases} 1 + OPT[W \setminus S_{j+1}, j, f_{x-1}] & \text{if } f = 1 \\ \min \left\{ OPT[W, j, f_{x-1}], 1 + OPT[W \setminus S_{j+1}, j, f_{x-1}] \right\} & \text{if } f = 0 \end{cases}$$

When S_{j+1} is the first element of the block B_x and the flag corresponding to B_x f is 1, then the set S_{j+1} has to be in the optimal solution as it is the only set among $\{S_1, \dots, S_j\}$ present in B_x . Since the set S_j is in the previous block B_{x-1} , we recursively look at the optimal solution with the flag corresponding to the recursive instance set to f_{x-1} . This justifies the above recursive formulation.

We compute the subproblems in increasing order subsets $W \subseteq \mathcal{U}$ and for each W increasing order of j and f . The solution to the problem is computed at $OPT[\mathcal{U}, m, f_x]$ where f_x is the value for the block B_x containing S_m . The number of problems is $2^{|\mathcal{U}|+1} \cdot m$ and for each subproblem we spend $\mathcal{O}(|\mathcal{U}|)$ time. Hence the total running time is $\mathcal{O}(2^{|\mathcal{U}|} \cdot (n - |\mathcal{U}|) \cdot |\mathcal{U}|)$ which is $\mathcal{O}(2^{|\mathcal{U}|} \cdot n^2)$. \square

We solve DS-DISJOINTCLUSTER in $\mathcal{O}^*(2^{|S|})$ time as follows. We construct the SET-COVER WITH PARTITION instance from the DS-DISJOINTCLUSTER instance as discussed earlier. It can be easily seen that there exists a solution of size ℓ in DS-DISJOINTCLUSTER instance if and only if there exists a solution of size ℓ in SET-COVER WITH PARTITION instance. We then use the algorithm Lemma 7.2.1 for SET-COVER WITH PARTITION.

In the algorithm for DS-CVD, for each guess $S' \subseteq S$ with $|S'| = i$, we construct the DS-DISJOINTCLUSTER instance with $|S| = k - i$ and solve it with running time $\mathcal{O}^*(2^{k-i})$. Hence the total running time is $\sum_{i=1}^k \binom{k}{i} \mathcal{O}^*(2^{k-i})$ which is $3^k n^{\mathcal{O}(1)}$.

We show that with some careful modifications to the above dynamic programming algorithm, we can obtain efficient FPT algorithms for minimum efficient, independent, total and threshold dominating set when parameterized by the size of cluster deletion set to show the following.

Theorem 32. EDS-CVD, TDS-CVD and IDS-CVD can be solved in $\mathcal{O}^*(3^k)$ time. THDS-CVD can be solved in $\mathcal{O}^*((r+2)^k)$ time.

Proof. We first give the algorithm for EDS-CVD.

Algorithm for EDS-CVD

Like in DS-CVD, we make a guess S' from the modulator. It is required all the vertices in S' should have a disjoint closed neighborhood as otherwise some vertex in S' is dominated twice. Otherwise, we return NO. We delete $N[S']$ and color the vertices in $N^2[S'] \setminus N[S']$ red to indicate that these vertices are to be dominated, but cannot be picked as picking one would make some vertex in $N[S']$ dominated twice. If some clique in $G \setminus S$ has all red vertices, we move on to make another guess from the modulator S' as these vertices in the clique cannot be dominated. If a clique contains both red and non-red vertices, we can delete the red vertices from that clique. Now from each clique in $G \setminus S$ exactly one vertex has to be picked. We are left to solve the following problem.

EDS-DISJOINTCLUSTER

Input: An undirected graph $G = (V, E), S \subseteq V(G)$ such that $G \setminus S$ is a cluster graph

Parameter: $|S|$

Question: Is there an efficient dominating set in G that is disjoint from S ?

For the problem we started with, the set S in this new formulation is the remaining vertices of S after deleting S' and $N[S']$ and the cluster graph is the union of original cliques after deleting the red vertices of the cliques.

We now give an algorithm to solve EDS-DISJOINTCLUSTER.

Any solution must pick exactly one vertex from each clique as one vertex is sufficient and necessary to dominate all vertices in a clique and we expect them to dominate S as well. Like we did in DS-CVD, we construct the set cover instance $(\mathcal{U}, \mathcal{F}, \mathcal{B})$. Here all the vertices of S must be exactly dominated once using the remaining vertices in $V \setminus S$. This suggests that the problem could be formulated as a variant of the EXACT SET COVER problem where you require every element in the universe to be covered exactly once. The additional requirement of picking exactly one set from each block leads to a reduction to the following problem.

EXACT SET-COVER WITH PARTITION

Input: A universe \mathcal{U} , a family of sets $\mathcal{F} = \{S_1, \dots, S_m\}$, a partition $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_q)$ of \mathcal{F} and an integer ℓ .

Parameter: $|\mathcal{U}|$

Question: Does there exist a subset $\mathcal{F}' \subseteq \mathcal{F}$ of size at most ℓ such that every $u \in \mathcal{U}$ is covered exactly once and from each block \mathcal{B}_i exactly one set is picked?

It can be easily seen that there exists a solution of size ℓ for EDS-DISJOINTCLUSTER instance if and only if there exists a solution of size ℓ for EXACT SET-COVER WITH PARTITION instance. We now give a dynamic programming algorithm to solve EXACT SET-COVER WITH PARTITION. For every nonempty subset $W \subseteq \mathcal{U}$, for every $j \in [m]$ and flag $f = \{0, 1\}$, we define $OPT[W, j, f]$ as the cardinality of the minimum subset X of $\{S_1, \dots, S_j\}$ such that each element of W is covered exactly once and from each block \mathcal{B}_i , there is exactly one set in X except the block \mathcal{B}_x containing the set S_j where we reset the flag to f to indicate that exactly f sets are required in that block.

We have $OPT[W, 1, f] = 1$ if $W = S_1$ and $f = 1$, else $OPT[W, 1, f] = \infty$. To compute all the values of $OPT[W, j, f]$, we initially set all the remaining values to ∞ . We give the following recursive formulation for $OPT[W, j+1, f]$ with $j \geq 1$.

- Case 1 : S_{j+1} is not the first set in its block \mathcal{B}_x

$$OPT[W, j+1, f] = \begin{cases} OPT[W, j, f] & \text{if } S_{j+1} \not\subseteq W \text{ or } f = 0 \\ \min \{ OPT[W, j, f], 1 + OPT[W \setminus S_{j+1}, j, 0] \} & \text{otherwise} \end{cases}$$

- Case 2 : S_{j+1} is the first set in its block \mathcal{B}_x .

$$OPT[W, j+1, f] = \begin{cases} \infty & \text{if } S_{j+1} \not\subseteq W \text{ and } f = 1 \\ 1 + OPT[W \setminus S_{j+1}, j, 1] & \text{if } S_{j+1} \subseteq W \text{ and } f = 1 \\ OPT[W, j, 1] & \text{if } f = 0 \end{cases}$$

The idea is similar to the earlier recursive formula for dominating set. Checking whether $S_{j+1} \subseteq W$ before adding to the solution ensures that every element in the universe is covered exactly once. We compute the subproblems in increasing order subsets $W \subseteq \mathcal{U}$ and for each W increasing order of j and f . The solution to the problem is computed at $OPT[\mathcal{U}, m, 1]$.

Hence, we have a $\mathcal{O}^*(2^{|S|})$ algorithm for EDS-DISJOINTCLUSTER. If EDS-DISJOINTCLUSTER returns ∞ for all choices of S' , we return NO as there is no such EDS in the graph. Since the number of subproblems remains the same as DS-CVD, the total running time to solve EDS-CVD is $\mathcal{O}^*(3^k)$.

Algorithm for IDS-CVD

The idea remains almost the same as in DS-CVD. For the guess in the modulator $S' \subseteq S$, the graph $G[S']$ has to be independent. If S' dominates all the vertices of a clique we can delete the clique as you cannot pick any vertex from this clique preserving independence. In the graph obtained after deleting $N[S'] \cap S$, we have to pick exactly one vertex from each clique to dominate vertices in $S \setminus N[S']$. Hence the IDS-CVD instance can be reduced to the SET-COVER WITH PARTITION problem instance with a slight modification where

instead of *at least* picking one set from each block we have to pick *exactly* one set. A similar dynamic programming algorithm can solve this problem giving us an overall running time of $\mathcal{O}^*(3^k)$.

Algorithm for THDS-CVD

Again our FPT algorithm starts with making a guess S' for the solution's intersection with S . We delete vertices in $S \cap N[S']$ that has r neighbours to S' . For the rest of the vertices $v \in V(G)$, we associate a weight $w(v) \in \{0, 1, \dots, r\}$ denoting the remaining number of times v is required to be dominated after S' is added to the r -threshold dominating set. Now we are left to solve the following problem (for each guesses S').

THDS-DISJOINTCLUSTER

Input: An undirected graph $G = (V, E)$, $S \subseteq V(G)$ such that $G \setminus S$ is a cluster graph, weight function $w : V \rightarrow \{0, 1, \dots, r\}$.

Parameter: $|S|$

Question: Is there a subset $D \subseteq V \setminus S$ of size l in G such that every vertex $v \in V$ has at least $w(v)$ neighbours in D ?

We now give an algorithm to solve THDS-DISJOINTCLUSTER. For each clique C_i , we have to pick at least $\max_{v_j \in C_i} w(v_j) + 1$ vertices to dominate all the vertices in the clique at least r times. This can be viewed as a weight corresponding to the clique.

Again we construct a SET COVER instance $(\mathcal{U}, \mathcal{F}, \mathcal{B})$ as done in DS-CVD. Since elements in \mathcal{U} are to be covered multiple times, the problem to be reduced is a variant of WEIGHTED MULTICOVER problem. The additional covering requirement in each of the blocks leads us to the following problem definition.

WEIGHTED SET-MULTICOVER WITH PARTITION

Input: A universe \mathcal{U} , a family of sets $\mathcal{F} = \{S_1, \dots, S_m\}$, a partition $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_q)$ of \mathcal{F} , a weight functions $w_{|\mathcal{U}\mathcal{U}} : \mathcal{U} \rightarrow [r]$, $w_{\mathcal{B}} : \mathcal{B} \rightarrow \{0, 1, \dots, r\}$ and an integer ℓ .

Parameter: $|\mathcal{U}|$

Question: Does there exist a subset $\mathcal{F}' \subseteq \mathcal{F}$ of size ℓ such that every $u \in \mathcal{U}$ is covered at least $w_{\mathcal{U}}(u)$ times and from each block \mathcal{B}_i at least $w_{\mathcal{B}}(\mathcal{B}_i)$ sets are picked?

It can be easily seen that there exists a solution of size ℓ for the THDS-DISJOINTCLUSTER instance if and only if there exists a solution of size ℓ for the WEIGHTED SET-MULTICOVER WITH PARTITION instance. We now give a dynamic programming algorithm to solve WEIGHTED SET-MULTICOVER WITH PARTITION. For every $j \in [m]$, for every weight vector $w = (w_{u_1}, w_{u_2}, \dots, w_{u_{|\mathcal{U}|}})$ (where $u_i \in \mathcal{U}$, $w_{u_i} \in \{0, 1, \dots, r\}$) and flag $f = \{0, 1, \dots, r\}$, we define $OPT[j, w, f]$ as the cardinality of the minimum subfamily X of $\{S_1, \dots, S_j\}$ such that each element $u_i \in \mathcal{U}$ is covered at least w_{u_i} times (note that $w_{u_i} = 0$ indicates that you do not need to cover u_i) and from each block \mathcal{B}_i , there is at least $w_{\mathcal{B}}(\mathcal{B}_i)$ sets in X except the block \mathcal{B}_x containing the set S_j where we reset the weight to f to indicate that at least f sets are required in that block.

We have $OPT[1, w, f] = 1$ if S_1 covers every element in $u_i \in U$ at least w_{u_i} times and $f \leq 1$, else $OPT[1, w, f] = \infty$. To compute all the values of $OPT[j, w, f]$, we initially set all the remaining values to ∞ and give the following recursive formulation for $OPT[j+1, w, f]$ with $j \geq 1$.

- Case 1 : S_{j+1} is not the first set in its block \mathcal{B}_x .

$$OPT[j+1, w, f] = \min \left\{ OPT[j, w, f], 1 + OPT[j, w', \max\{f-1, 0\}] \right\}$$

where w' is the weight-vector after subtracting 1 from w_{u_i} for each of the elements $u_i \in S_{j+1}$ where $w_{u_i} > 0$.

- Case 2 : S_{j+1} is the first set in its block \mathcal{B}_x .

$$OPT[j+1, w, f] = \begin{cases} 1 + OPT[j, w', w_{\mathcal{B}}(\mathcal{B}_{x-1})] & \text{if } f = 1 \\ \min \left\{ OPT[j, w, w_{\mathcal{B}}(\mathcal{B}_{x-1})], \right. \\ \left. 1 + OPT[j, w', w_{\mathcal{B}}(\mathcal{B}_{x-1})] \right\} & \text{if } f = 0 \\ \infty & \text{otherwise} \end{cases}$$

where w' is the weight-vector after subtracting 1 from w_{u_i} for each of the elements $u_i \in S_{j+1}$ where $w_{u_i} > 0$.

Again the idea is similar to the recursive formulation of DS-CVD with the choice of whether S_{j+1} is in the optimal solution or not. When S_{j+1} is in the optimal solution, we decrease the weight requirements of the elements in the set by one and get the new weight vector w' . When S_{j+1} is the first set in the block, in the recursive subproblem, we set the flag to $w_{\mathcal{B}}(\mathcal{B}_{x-1})$ corresponding to the block \mathcal{B}_{x-1} that contains the set S_j .

We compute the subproblems in increasing order of j , w and f . The solution to the problem is computed at $OPT[m, w_1, w_{\mathcal{B}}(\mathcal{B}_x)]$ where w_1 corresponds to the weight-vector from the input function $w_{\mathcal{U}}$ and $w_{\mathcal{B}}(\mathcal{B}_x)$ is the weight of the block \mathcal{B}_x containing S_m . The number of problems are $m \cdot (r+1)^{|\mathcal{U}|} \cdot (r+1)$ and for each subproblem we take $\mathcal{O}(|\mathcal{U}|)$ time to update the weight vector. The total running time is $\mathcal{O}((n - |S|) \cdot (r+1)^{|S|} \cdot |S|) = \mathcal{O}^*((r+1)^{|S|})$.

Now for each guess S' in the modulator with $|S'| = i$, we get a

THDS-DISJOINTCLUSTER instance with $|S| = k - i$ and solve it with running time $\mathcal{O}^*((r+1)^{k-i})$. Hence the total running time in solving THDS-CVD is $\sum_{i=1}^k \binom{k}{i} \mathcal{O}^*((r+1)^{k-i}) = \mathcal{O}^*((r+2)^k)$.

Since TDS-CVD is THDS-CVD with $r = 1$, we have a $\mathcal{O}^*(3^k)$ algorithm to solve TDS-CVD. □

7.2.2 Lower bounds

We first give lower bounds for DS-CVD and TDS-CVD by giving a reduction from the SET COVER problem.

Lemma 7.2.2. *There is a polynomial time algorithm that takes an instance $(\mathcal{U}, \mathcal{F}, \ell)$ of SET COVER with $\ell > 1$ and outputs an instance (G, ℓ) of DS-CVD (or TDS-CVD) such that G has a cluster vertex deletion set with exactly $|\mathcal{U}|$ vertices, such that $(\mathcal{U}, \mathcal{F}, \ell)$ has a set cover of size ℓ if and only if G has a (total) dominating set of size ℓ .*

Proof. Consider a SET COVER instance $(\mathcal{U}, \mathcal{F}, \ell)$ with $\mathcal{U} = \{u_1, \dots, u_k\}$ and $\mathcal{F} = \{S_1, \dots, S_m\}$. Construct the graph $G = (U \uplus V, E)$ with vertex sets $U = \{u_1, \dots, u_k\}$ and $V = \{s_1, \dots, s_m\}$. Every vertex in U corresponds to an element in the universe \mathcal{U} and every vertex $s_j \in V$ corresponds to the set $S_j \in \mathcal{F}$ for $j \in [m]$. We add edges (u_i, s_j) if $u_i \in S_j$. We also add edges (s_i, s_j) for all $i \neq j$ to make $G[V]$ a clique with m vertices. Hence the graph $G \setminus U$ is a cluster graph with $|U| = k$.

We now claim that there is a subset $\mathcal{F}' \subseteq \mathcal{F}$ of size ℓ covering \mathcal{U} if and only if and only if there is a dominating set of size ℓ in G .

We first prove the forward direction. We claim that the set of vertices in V of size ℓ corresponding to the subsets in \mathcal{F}' forms a dominating set. More specifically, let $V' = \{s_i \in V \mid S_i \in \mathcal{F}'\}$. Since $G[V]$ is a clique, by picking at least one vertex from V all the vertices of V are dominated. Since \mathcal{F}' covers all the elements of \mathcal{U} , for each vertex $u_i \in U$ there exists some $s_j \in V$ such that there is an edge (u_i, s_j) in G . Hence all the vertices of U are dominated as well. Hence V' is a dominating set in G of size ℓ .

We now prove the reverse direction. First, we claim that there is a dominating set of size ℓ which does not contain any vertex from U . Let $D \subseteq V(G)$ be a dominating set of size ℓ in G . Suppose there is a vertex $u_i \in U$ in D . Since $G[U]$ is an independent set, u_i can only cover some subset of vertices in V other than u_i . But by picking any vertex of V

adjacent to u_i , all the vertices of V can also be covered other than u_i . Hence $D \setminus \{u_i\} \cup \{v\}$ for any neighbour v of u_i (such a neighbour exists as otherwise u_i is not present in any of the sets) is also a dominating set of the same size. We do this for all vertices $u \in U$ and get a dominating set D of size at most ℓ not containing any vertex in U . By picking the sets corresponding to the vertices in $D \subseteq V$, we get a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ of size ℓ covering \mathcal{U} .

Note that the dominating set in the above claim is also a total dominating set for $\ell > 1$ where at least two vertices of the clique are in the solution. This proves the lemma. \square

The following theorem follows from the above lemma and Set Cover Conjecture (Conjecture 2.2.3).

Theorem 33. *DS-CVD and TDS-CVD cannot be solved in $\mathcal{O}^*((2 - \varepsilon)^k)$ running time for any $\varepsilon > 0$ unless Set Cover Conjecture fails.*

Proof. Suppose that there is an algorithm solving DS-CVD in $\mathcal{O}^*((2 - \varepsilon)^k)$ running time. Then by Lemma 7.2.2, we can solve SET COVER with $|\mathcal{U}| = k$ in running time $\mathcal{O}^*((2 - \varepsilon)^k)$ violating the Set Cover Conjecture (SCC). This completes the proof. Since the dominating set in the reduction is also total, TDS-CVD also cannot be solved in $\mathcal{O}^*((2 - \varepsilon)^k)$ running time for any $\varepsilon > 0$ as well. \square

The following theorem follows from Theorem 2 and Lemma 7.2.2.

Theorem 34. *DS-CVD, TDS-CVD and THDS-CVD do not have polynomial sized kernels unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.*

Proof. SET COVER parameterized by the size of the universe does not admit polynomial sized kernels unless the polynomial hierarchy collapses to the third level, i.e. $\text{NP} \subseteq \text{coNP}/\text{poly}$ [56]. Since the reduction provided in Lemma 7.2.2 is a polynomial parameter transformation (PPT) (see Definition 2.2.7), by Property 2.2.1, we have that these problems including THDS-CVD do not have polynomial kernel unless $\text{NP} \subseteq \text{coNP}/\text{poly}$. \square

Note that the proof idea of Theorem 34 does not work for IDS-CVD. This is because, in the reduction, we turn the graph induced by the family of sets into a clique. Hence, only one vertex of the clique can be in an independent dominating set of such a graph. Nevertheless, we give a $\mathcal{O}^*((2 - \varepsilon)^k)$ lower bound for IDS-CVD under SETH using the following lower bound result for a different problem which is MMVC-VC.

Theorem 35 ([154]). *Unless SETH fails, MMVC-VC cannot be solved in $\mathcal{O}^*((2 - \varepsilon)^k)$ time. Moreover, MMVC-VC does not admit polynomial sized kernel unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.*

We now prove the following observation from which it follows that the complement of a maximum minimal vertex cover is a minimum independent dominating set. This observation is already known from before [24, 154]. But we still give proof for completeness.

Observation 7.2.1. *If T is a minimal vertex cover of the graph G , then $V(G) \setminus T$ is an independent dominating set in G . Furthermore, if T is a maximum sized minimal vertex cover, then $V(G) \setminus T$ is a minimum sized independent dominating set.*

Proof. A set $T \subseteq V(G)$ is a maximum minimal vertex cover when T is a minimal vertex cover and among all minimal vertex covers, T has the maximum number of vertices. Let $D = V(G) \setminus T$. Clearly, D is an independent set. Note that for all $u \in V(G)$, either $u \in D$ or if $u \notin D$, then $u \in T$. As T is minimal, there must be a neighbor v that is not in T . So, such a neighbor can only be in D . So, D is a dominating set. Hence D is an independent dominating set. Now suppose that D is not a minimum independent dominating set. Then there exists another independent dominating set D' such that $|D'| < |D|$. Now consider $T' = V(G) \setminus D'$. Clearly $|T'| > |T|$ and any vertex in D' has some neighbor in T' (otherwise D' is not minimum). So, T' is a minimal vertex cover. But then T is not a maximum minimal vertex cover and that is a contradiction. So, D which is $V(G) \setminus T$ is a minimum independent dominating set. \square

It follows from the observation that the MMVC-VC problem is equivalent to IDS-VC. From Theorem 35, we have the following result.

Theorem 36. *IDS-VC cannot be solved in $(2 - \varepsilon)^k n^{\mathcal{O}(1)}$ time unless SETH fails. Moreover IDS-VC does not have any polynomial kernel unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.*

We now note that any vertex cover is a cluster vertex deletion set. Hence the cluster vertex deletion set size parameter is at most the vertex cover size parameter. From Theorem 36, we have the following result.

Corollary 7.2.1. *IDS-CVD cannot be solved in $O^*((2 - \varepsilon)^k)$ time for any $\varepsilon > 0$ unless SETH fails. Moreover IDS-CVD does not have any polynomial kernel unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.*

For EDS-CVD, we can only prove a weaker lower bound of $2^{o(k)}$ time assuming ETH. But we give the lower bound for EDS parameterized by even a larger parameter, the size of a vertex cover. We have the following result.

Theorem 37. *EDS-VC cannot be solved in $2^{o(|S|)}$ time unless ETH fails.*

Proof. We give a reduction from 3-SAT to EDS-VC.

Construction: Let ϕ be the 3-SAT input formula with variables x_1, \dots, x_n and clauses C_1, \dots, C_m . For all $i \in [n]$, we create vertices v_i and \bar{v}_i . The vertex v_i corresponds to x_i appearing in its pure form and \bar{v}_i corresponds to x_i appearing in negated form. For all $i \in [n]$, we add edge (v_i, \bar{v}_i) . We call this as variable gadget. Let C be a clause. We create vertices $c_1, c_2, c_3, d_0, d_1, d_2, d_3, d_{1,2}, d_{2,3}, d_{1,3}$ and form a clause gadget as in Figure 7.1. c_1, c_2, c_3 represents a copy of clause C . We repeat this process for each of the clauses. Suppose a clause $C = (v_i \vee \bar{v}_j \vee v_p)$. We add edges $(v_i, c_1), (\bar{v}_j, c_2), (v_p, c_3)$. This completes our construction.

Now we show that ϕ is satisfiable if and only if G_ϕ has an EDS of size $n + m$.

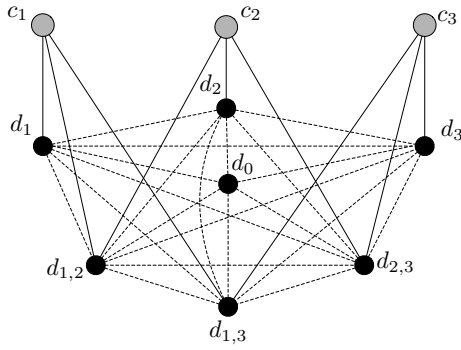


Figure 7.1: An illustration of the construction described in Theorem 37

To prove the forward direction, let ϕ be satisfiable with $\bar{a} = (a_1, \dots, a_n)$ being a satisfying assignment where $\forall i \in [n] : a_i \in \{true, false\}$. We construct an efficient dominating set D as follows. If $a_i = true$, then we add v_i into D . Otherwise we add \bar{v}_i into D . Fix a clause C . If only c_1 is dominated, then we pick $d_{2,3}$ into D . The case is symmetric when only c_2 or c_3 is dominated. Otherwise if c_1 and c_2 are dominated but c_3 is not dominated, then we add d_3 into D . Other cases are symmetric when exactly two from c_1, c_2, c_3 are dominated. When all c_1, c_2, c_3 are dominated, then we pick d_0 . In this way, we pick exactly one vertex from each of the clause gadgets and get an efficient dominating set of size $n + m$.

We now prove the reverse direction. Let D be an efficient dominating set size $n + m$. Clearly by construction exactly one vertex is picked from each of the clause gadgets and exactly one vertex is picked from each of the variable gadgets. We construct an assignment $\bar{a} = (a_1, \dots, a_n)$ as follows. If $v_i \in D$, then we assign $a_i = true$, otherwise $\bar{v}_i \in D$ and we assign $a_i = false$. Suppose \bar{a} does not satisfy ϕ . Then there exists a clause C that is not satisfied by \bar{a} . It means that none of c_1, c_2, c_3 is dominated by variable vertices. Now note that to dominate them at least two vertices are required from the clause gadget. But between every two vertices in the clause gadget, the distance is two. So, in such a case D was not an efficient dominating set which is a contradiction. So, \bar{a} satisfies ϕ proving ϕ to be a satisfiable formula.

We have proved a reduction from 3SAT to EDS-VC. Suppose there exists algorithm \mathcal{B} that solves EDS-VC in $\mathcal{O}^*(2^{o(k)})$ time. Now we use algorithm \mathcal{B} to provide an algorithm

for 3-SAT running in $\mathcal{O}^*(2^{o(n+m)})$ time. Let ϕ be an instance of 3SAT consisting of n variables and m clauses. We construct G_ϕ as the construction described above. Now we use algorithm \mathcal{B} . If \mathcal{B} outputs $(G_\phi, n+m)$ as NO-instance then we output that ϕ is unsatisfiable. Otherwise \mathcal{B} outputs an efficient dominating set of size $n+m$. By construction as described above, we construct an assignment $\bar{a} = (a_1, \dots, a_n)$ for 3SAT and output as the satisfying assignment. As \mathcal{B} runs in $\mathcal{O}^*(2^{o(n+m)})$ time and the transformation from ϕ to G_ϕ takes polynomial time, we get an algorithm for 3SAT in $\mathcal{O}^*(2^{o(n+m)})$ time. From [88, 87] (See also Theorem 14.4 of [45]) we know that unless ETH fails, 3-SAT cannot be solved in $\mathcal{O}(2^{o(n+m)})$ time. This contradicts ETH. So EDS-VC cannot be solved in $\mathcal{O}^*(2^{o(|S|)})$ time unless ETH fails. Now, we know that any vertex cover is a cluster vertex deletion set. So, EDS-CVD also cannot be solved in $\mathcal{O}^*(2^{o(k)})$ time unless ETH fails. \square

Since cluster vertex deletion size is at most the vertex cover size parameter, we have the following corollary.

Corollary 7.2.2. *EDS-CVD cannot be solved in $2^{o(|S|)}$ time unless ETH fails.*

7.3 Dominating Set variants parameterized by SVD size

In this section, we address the parameterized complexity of dominating set variants when parameterized by the size of a given SVD set S . Note that DS and TDS are NP-hard on split graphs [145]. Hence we focus only on EDS and IDS.

We assume that S is given with the input. Otherwise given (G, k) , we use an $\mathcal{O}^*(1.27^{k+o(k)})$ algorithm due to Cygan and Pilipczuk [46] to find a set of vertices of size at most k whose removal makes G into a split graph.

7.3.1 EDS and IDS parameterized by SVD size

We provide a simple algorithm for IDS-SVD. The idea is to make a guess for the solution within the SVD and solve the resulting disjoint problem in polynomial time. It turns out that it works for EDS-SVD too.

Theorem 38. *EDS-SVD and IDS-SVD can be solved in $O^*(2^k)$ time.*

Proof. We will essentially prove that once we guess a correct subset S' of S that is in the EDS solution we seek, the remaining set of vertices can be determined in polynomial time, and then the claim will follow as we will try all possible guesses of S' . As in the case of EDS parameterized by cluster vertex deletion set, once we guess the subset S' , we delete $N[S']$ and mark $N^2[S']$ red. Let $S'' = S \setminus N[S']$ be the remaining vertices of S . Let (C, I) the partition of $G \setminus S$ into a clique C and an independent set I .

To dominate the vertices of C , we need to pick some non-red vertex of C or a non-red vertex from I . In particular, there are up to $|C| + 1$ choices (at most one vertex from C) of vertices to be picked from C . For each such choice, as before, we delete the closed neighbors of the vertex picked, and move the vertices in the second neighborhood to S'' . If we decide to pick no vertex from C , we move the vertices of C to S'' . After these choices have been made, all vertices of C have been deleted (or moved to S''). Now if there are any red vertices in I , we move to the new guess, as such vertices cannot be dominated. Otherwise, to dominate the remaining non-red vertices in I , we need to pick them all. Now we check whether the final solution picked is an EDS for the entire graph (in particular they should uniquely dominate S''). This proves that EDS-SVD can be solved in $O^*(2^k)$ time.

The algorithm for INDEPENDENT DOMINATING SET also works similarly. First, we guess an independent set $S' \subseteq S$. We delete $N[S']$ from G . Now we are left with the split graph (C, I) and vertices in $T = S \setminus N[S']$. We have to use vertices from $C \cup I$ only to dominate vertices in $C \cup T \cup I$. We guess vertices in C . There can be at most $|C| + 1$ many guesses

since at most one vertex can be part of the solution. If $v \in C$ is decided to be picked in the solution, then $N[v]$ is deleted. Now $I \setminus N[v]$ is essential to be part of the solution. If $A = S' \cup \{v\} \cup (I \setminus N[v])$ forms an independent dominating set, then we store A as a candidate for being a solution. If no vertex from C is decided to be picked into the solution, then we have to pick all vertices from I into the solution. If $S' \cup I$ is a solution, then we store $S' \cup I$ also as a candidate for being a solution. We go through all these candidates and choose one that is of the smallest cardinality. We repeat this step for all possible subsets of S that forms an independent set. So, for IDS-SVD also, there exists an algorithm running in time $\mathcal{O}^*(2^{|S|})$ for IDS-SVD. \square

7.3.2 Improved Algorithm for EDS-SVD

In this section, we give an improved algorithm for EDS-SVD parameterized by the size of a given split vertex deletion set S breaking the barrier of $\mathcal{O}^*(2^k)$.

Let $F = G \setminus S$. As F is a split graph, $V(F) = C \uplus I$ where C induces a clique and I induces an independent set. The algorithm uses the standard branching technique. Consider any efficient dominating set D of a graph. Any two vertices $u, v \in D$ must have a distance of at least three. At any intermediate stage of the algorithm, we make a choice of not picking a vertex and we mark such vertices by coloring them red. Other vertices are colored blue. Hence all vertices of G are blue initially.

We initialize $D = \emptyset$ which is the solution set we seek. Consider any pair of blue vertices $x, y \in S$. If the distance between x and y is at most two in G , then we use the following branching rule.

Branching Rule 7. *Consider a pair of blue vertices $x, y \in S$ such that the distance between x and y is at most two in G . In the first branch, we add x into D , delete $N[x]$ from G , color the vertices in $N^{\leq 2}(x)$ by red. In the second branch, we add y into D , delete $N[y]$ from G , color the vertices in $N^{\leq 2}(y)$ by red. In the third branch, we color x, y by red. (See*

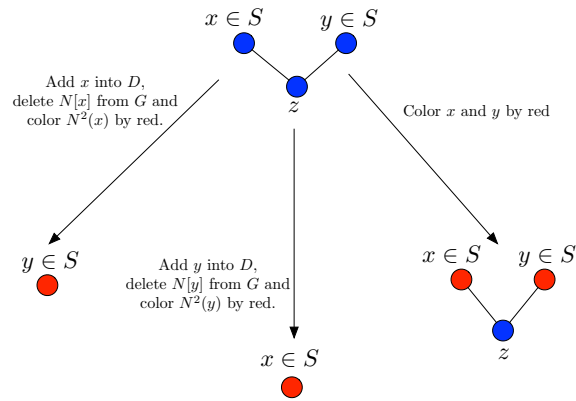


Figure 7.2: Illustration of Branching Rule 7. Note that the number of blue vertices drop by at least two in each of the branches.

Figure 7.2 for an illustration.)

Clearly, the branches are exhaustive as both x and y cannot be in the EDS solution we seek.

We measure the progress of the algorithm by $\mu(G)$ which is the number of blue vertices in S , which is k initially. In the first branch, x is deleted from S and y is colored red. Symmetrically in the second branch, y is deleted from S and x is colored red. In the third branch, x and y are colored red. So in all the branches, $\mu(G)$ drops by at least two resulting in a $(2,2,2)$ branching rule.

When this branching rule is not applicable, for every pair of blue vertices $x, y \in S$, $N[x] \cap N[y] = \emptyset$. Now, as C is a clique, we can have at most one vertex from C in the solution. When we decide to pick some vertex $v \in C$ into the solution, then we delete $N[v]$ and color $N^2(v)$ as red. So all vertices of C get deleted. There are at most $|C|$ vertices in C . When we decide not to pick any vertex from C into the solution, then we color all vertices of C as red. So we have $(|C| + 1)$ choices from the vertices of C . Measure $\mu(G)$ does not increase in any of these choices. A multiplicative factor of $(|C| + 1)$ would come in the running time because of this one-time branching. Now, we are left with only the vertices of I . We apply the following reduction rule to rule out some simple boundary conditions.

Reduction Rule 4. *If there exists a red vertex $x \in V(G)$ such that $N_G(x)$ has only one blue*

vertex y , then add y into D , delete $N[y]$ from G and color $N^{=2}(y)$ as red. Also if there exists a blue vertex $x \in V(G)$ such that $N_G(x)$ contains no blue vertex, then add x into D , delete $N[x]$ from G and color $N^{=2}(x)$ as red.

It is easy to see that the above reduction rule is safe. Note that we have some blue vertices in I . Such vertices can only be dominated by themselves or a unique blue vertex in S , as otherwise Branching Rule 7 would have been applicable. Now, suppose that there exists a blue vertex $x \in S$ that has at least two blue neighbors $u, v \in I$. If we decide to pick u (or symmetrically v) into D , then we are not allowed to pick x or v (symmetrically u) in D but then u or v cannot be dominated. This forces x into D . We have the following reduction rule.

Reduction Rule 5. *If there exists a blue vertex $x \in S$ such that $N_G(x)$ contains at least two blue neighbors in I , then add x into D , delete $N[x]$ from G and color vertices in $N^{=2}(x)$ red.*

Lemma 7.3.1. *Reduction Rule 5 is safe.*

Proof. The safeness of this reduction rule is based on the fact that any feasible solution (if exists) must contain x under this construction. Suppose not. Then let D be an efficient dominating set that does not contain x . Then either $u \in I$ or $v \in I$ but not both. Note that any blue vertex in I has only one blue neighbor in S , as all vertices of C are red. So if $u \in D$ then $x, v \notin D$. Then v cannot be dominated at all. Similarly if $v \in D$, then u cannot be dominated at all. So $x \in D$ and this concludes the proof. \square

Lemma 7.3.2. *Reduction Rules 4 and 5 do not increase $\mu(G)$.*

Proof. Reduction Rule 4 does not add any blue vertex into S , rather can delete blue vertex from S . Similarly, Reduction Rule 5 only deletes a blue vertex from S . So, μ does not increase in either of these two reduction rules. \square

Now if there are red vertices in I having no blue neighbor in S , then we move to the next branch as such a vertex cannot be dominated. Thus any blue vertex in I has only one blue neighbor in S and any blue vertex in S has only one blue neighbor in I . As Reduction Rule 4 is not applicable, any red vertex $x \in S \cup C$ has at least two blue neighbors in $u, v \in N_G(x)$. Clearly both $\{u, v\} \not\subset S$ as otherwise Branching Rule 7 would have been applicable. So, now we are left with the case that $u, v \in I$ or $u \in I, v \in S$ but (u, v) may or may not be an edge. Now we apply the following branching rule.

Branching Rule 8. *Let x be a red vertex in S with two blue neighbors u, v .*

1. *If $u, v \in I$, then we branch as follows. In one branch we add u into D , delete $N[u]$ from G , color $N^{=2}(u)$ as red. As $v \in N^{=2}(u)$ and v has only one blue neighbor $z \in S$, we add z also into D , delete $N[z]$ from G and color $N^{=2}(z)$ by red. In the second branch, we add v into D , delete $N[v]$ from G , color $N^{=2}(v)$ as red. As $u \in N^{=2}(v)$ and u has only one blue neighbor $y \in S$, we add y also into D , delete $N^{=2}(y)$ from G and color $N^{=2}(z)$ by red. In the third branch, color both u and v by red. Add the only blue neighbor y of u and z of v into D . Delete $N[y], N[z]$ from G and color the vertices in $N^{=2}(y) \cup N^{=2}(z)$ by red.*
2. *$u \in I, v \in S, (u, v) \notin E(G)$, then we branch as follows. In the first branch, we add u to D , color v as red. This forces us to pick the only blue neighbor z of v where $z \in I$. So, we add z to D . Delete $N[u], N[z]$ from G and color $N^{=2}(u), N^{=2}(z)$ as red. In the second branch, we color u as red. This forces us to pick the only neighbor y of u where $y \in S$. And we pick v into D as well as y into D . We delete $N[v], N[y]$ from G and color $N^{=2}(v), N^{=2}(y)$ by red. In the third branch, we color both u and v by red. This forces us to pick the only blue neighbor $z \in N_G(v) \cap I, y \in N_G(u) \cap S$ into D . So, we pick z into D , delete $N[z], N[y]$ from G and color $N^{=2}(y), N^{=2}(z)$ by red. (See Figures 7.3 and 7.4 for detailed illustrations.)*

It is easy to see that $\mu(G)$ drops by at least two in all three branches as eventually two blue vertices of S get deleted in all the branches.

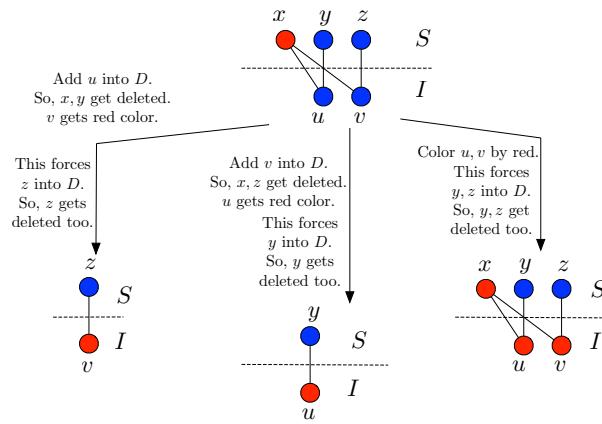


Figure 7.3: Illustration of Branching Rule 8 for the first case. Note that the number of blue vertices in S drops by at least two in each of the branches.

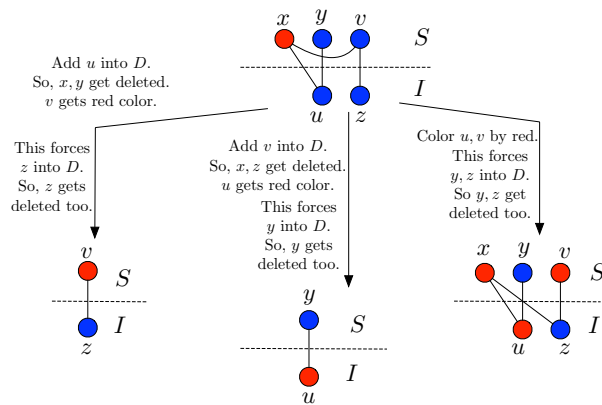


Figure 7.4: Illustration of Branching Rule 8 for the second case. Note that the number of blue vertices in S drops by two in each of the branches.

When none of the above rules are applicable, then we have $u \in S, v \in I$ and $(u, v) \in E(G)$. We know that either $u \in D$ or $v \in D$. Consider the red vertices in $N(u)$ and red vertices in $N(v)$. As Branching Rule 7, Reduction Rule 4 and Branching Rule 8 are not applicable, by the following lemma using which we can pick u or v arbitrarily.

Lemma 7.3.3. *If Branching Rule 7, Reduction Rule 4 and Branching Rule 8 are not applicable, then $N(u) \setminus \{v\} = N(v) \setminus \{u\}$.*

Proof. Suppose $x \in N(u) \setminus \{v\}$. Clearly x is a red vertex by the premise. As Branching Rule 7 is not applicable, x cannot have any other neighbor which is a blue vertex of S . As Reduction Rule 4 is not applicable, x has another blue neighbor let y . And $y \in I$. If $(u, y) \notin E(G)$, then Branching Rule 8 is applicable. So $(u, y) \in E(G)$ implying that $y = v$. So, $x \in N(v) \setminus \{u\}$ implying that $N(u) \setminus \{v\} \subseteq N(v) \setminus \{u\}$. Similarly we can prove that $N(v) \setminus \{u\} \subseteq N(u) \setminus \{v\}$. This completes the proof of the above lemma. \square

This completes the description of our algorithm that consists of a sequence of reduction rules and branching rules. The measure is k initially and the branching continues as long as k drops to 0. So, we have the following recurrence.

$$T(k) \leq 3T(k-2) + \alpha \cdot (n+k)^c$$

Solving this recurrence, we get $1.732^k \cdot n^{\mathcal{O}(1)}$ implying the following theorem.

Theorem 39. *EDS-SVD can be solved in $\mathcal{O}^*(3^{k/2})$ time.*

7.3.3 Lower Bounds for IDS and EDS

We know that any vertex cover is a split vertex deletion set. So, we have the following corollary as a consequence of Theorem 35.

Corollary 7.3.1. *IDS-SVD cannot be solved in $\mathcal{O}^*((2-\epsilon)^k)$ time unless SETH fails and it does not admit polynomial kernels unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.*

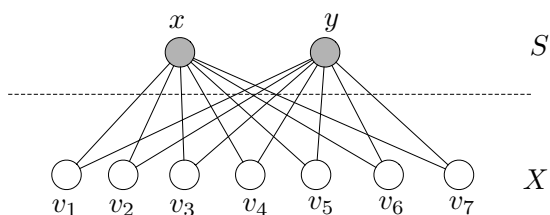


Figure 7.5: In this example, S is a minimum vertex cover of this graph. But there is no minimal vertex cover that contains only x , but not y from S .

For EFFICIENT DOMINATING SET as the size of the SVD set is always smaller than the size of the vertex cover, we have the following corollary of Theorem 37.

Corollary 7.3.2. *EDS-SVD cannot be solved in $2^{o(|S|)}$ time unless ETH fails.*

7.4 Concluding Remarks

We have initiated a study of structural parameterizations of some dominating set variants and complemented them with lower bounds based on ETH and SETH. One immediate open problem is to narrow the gap between upper and lower bounds, especially for the dominating set variants parameterized by the size of a cluster vertex deletion set.

We know that INDEPENDENT DOMINATING SET is the complementary version of MAXIMUM MINIMAL VERTEX COVER problem. So, one natural approach for an $\mathcal{O}^*(2^k)$ algorithm for IDS-CVD is to apply the ideas used in [154] to get $\mathcal{O}^*(2^k)$ algorithm for MMVC-VC. But this seems to require more work, as there may not exist a minimal vertex cover that intersects the CVD set S in a particular subset (See Figure 7.5).

Recently Bergougnoux et al. [15] have given an $\mathcal{O}^*(2^{O(k)})$ algorithm for connected dominating set (where we insist on the dominating set to induce a connected graph) for clique-width k graphs when the k -expression given as input. An interesting open problem is whether connected dominating set has a simpler FPT algorithm, as in the FPT algorithms in this chapter, when parameterized by cluster vertex deletion set.

Part III

Deletion with additional constraints

Chapter 8

Parameterized Complexity of Conflict-Free Set Cover

8.1 Introduction and Previous Work

SET COVER is one of the well-studied classical NP-hard problems. In the SET COVER problem, we have a universe \mathcal{U} , a family \mathcal{F} of subsets of \mathcal{U} and an integer k and the goal is to find a subfamily \mathcal{F}' of size at most k such that $\bigcup_{S \in \mathcal{F}'} S = \mathcal{U}$.

SET COVER is very well-studied in a variety of algorithmic settings, especially in the realm of approximation algorithms and parameterized complexity. Unfortunately, SET COVER when parameterized by solution size k is W[2]-hard [45] and hence is unlikely to be FPT.

It has been seen in computational problems [8, 7, 6, 13, 14, 90, 140, 99, 139, 53] where additional constraints are enforced on the solution we seek. One category of such problems is *choice* problems which can be described as follows. There is a set V from which we seek a solution subset. The set is partitioned into groups and the solution requires picking exactly one representative element from each of the groups. For example, consider the problem MULTICOLORED CLIQUE where the vertex set is partitioned into groups, and the

solution we seek is a clique such that exactly one vertex of the clique is present in each group. Another example in the geometric setting is by Arkin and Hassin [8] where they look at the following problem. Given a collection of points partitioned into groups and a matrix describing the distance between pairs of points, find a set of points such that exactly one point is in each of the groups and the set has minimum diameter.

We can generalize these choice problems to a setting where we say that some pairs of elements in the problem are in conflict with each other and hence cannot go in the solution together. This can be modeled by defining a graph on the elements and an edge (u, v) is added if elements u and v do not go into the solution together or in other words form a conflict. Hence a solution without conflicts will form an independent set in this graph. Looking back at the example of MULTICOLORED CLIQUE, we have a conflict-graph where each group forms a clique and there are no edges across any pair of groups.

Conflict-free versions of classical problems in P like MAXIMUM FLOW [139], MAXIMUM MATCHING [53], SHORTEST PATH [99], KNAPSACK [140], BIN PACKING [59] and SCHEDULING [60] have been studied. A study of some geometric problems in the conflict-free setting was initiated recently [7, 6, 13, 14] motivated by various applications. Conflict-free version of graph problems like VERTEX COVER [90], FEEDBACK VERTEX SET [1], SPLIT VERTEX DELETION [90] have been studied from the parameterized point of view very recently. Some of the problems above are covering problems. Since SET COVER is a very general covering problem, studying the conflict-free version of SET COVER contributes to advancing this framework.

We look at the conflict-free version of SET COVER defined as follows:

CONFLICT-FREE SET COVER

Input: An universe \mathcal{U} , a family \mathcal{F} of subsets of \mathcal{U} , a graph $G_{\mathcal{F}}$ with vertex set \mathcal{F} and an integer k .

Question: Is there a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ of size at most k such that $\cup_{F \in \mathcal{F}'} F = \mathcal{U}$ and \mathcal{F}' forms an independent set in $G_{\mathcal{F}}$?

Note that if $G_{\mathcal{F}}$ is edgeless, the problem is equivalent to SET COVER as every subset of vertices of $G_{\mathcal{F}}$ forms an independent set. Hence if $G_{\mathcal{F}}$ is from a graph class that contains edgeless graphs, when SET COVER is $W[2]$ -hard with respect to some parameter, then CONFLICT-FREE SET COVER is also $W[2]$ -hard with respect to the same parameter. Therefore, the only interesting cases of CONFLICT-FREE SET COVER are those special instances or parameterizations where SET COVER is FPT or when $G_{\mathcal{F}}$ is from a graph class that does not contain edgeless graphs.

One such example is SET COVER when parameterized by the size of the universe \mathcal{U} . There is a $2^{|\mathcal{U}|}(|\mathcal{U}| + |\mathcal{F}|)^{\mathcal{O}(1)}$ time algorithm for this problem using dynamic programming over subsets of \mathcal{U} [69]. Another example is a restricted version of SET COVER where every pair of sets in \mathcal{F} intersect in at most c elements for a constant c . This version of SET COVER which we call c -INTERSECTION SET COVER is known to be FPT parameterized by k and has a kernel of universe size ck^2 and $\binom{ck^2}{c+1}$ sets [145]. Problems like COVERING POINTS BY LINES can be seen as special cases of c -INTERSECTION SET COVER [108].

We note that like the SET COVER problem, CONFLICT-FREE SET COVER is trivially FPT parameterized by $|\mathcal{F}|$ due to the simple brute-force algorithm of choosing at most k sets from \mathcal{F} .

Unlike the SET COVER problem, in CONFLICT-FREE SET COVER duplicate sets do play an important role. This is because two identical sets in the family \mathcal{F} can have different neighborhood relations in the graph $G_{\mathcal{F}}$ which matters in the independence requirement of the solution. We study CONFLICT-FREE SET COVER both in the presence and absence of duplicate sets in \mathcal{F} . Note that if there are no duplicate sets in the family \mathcal{F} , $|\mathcal{F}| \leq 2^{|\mathcal{U}|}$.

Banik et al. [14] studied CONFLICT-FREE SET COVER in the context of some geometric covering problems having FPT algorithms. They showed that one of their geometric covering problems in the conflict-free setting is $W[1]$ -hard parameterized by solution size k when $G_{\mathcal{F}}$ is from those classes of graphs where INDEPENDENT SET is $W[1]$ -hard. They also showed that CONFLICT-FREE SET COVER is FPT parameterized by k whenever SET

$G_{\mathcal{F}}$ restriction	Solution size k	Universe size $ \mathcal{U} $
General graph	W[1]-hard [14]	$\binom{ \mathcal{F} }{ \mathcal{U} } \leq 2^{ \mathcal{U} ^2}$ algorithm without duplicates, $2^{ \mathcal{U} \log \mathcal{F} }$ ℓb (Theorem 44), W[1]-hard with duplicates (Lemma 42)
Empty graph	FPT* †	FPT †, npk †
Bipartite	$f(k) \mathcal{F} ^{o(k)}$ ℓb (Theorem 40)	W[1]-hard with duplicates (Lemma 42)
Chordal		FPT (Theorem 48), npk †
d -degenerate	FPT* [14]	FPT †, npk †
Nowhere Dense	FPT* (Theorem 46)	FPT (Theorem 46), npk †
Bounded #MIS	FPT* (Theorem 45)	FPT (Theorem 45), npk †

Table 8.1: Table of results: CONFLICT-FREE SET COVER . FPT* denotes that the problem is FPT whenever the SET COVER variant is FPT, #MIS abbreviates ‘number of maximal independent sets’, † denotes results from existing literature (other than those cited), ℓb abbreviates ‘lower bound’, npk abbreviates ‘no polynomial kernel’

COVER is FPT parameterized by k when $G_{\mathcal{F}}$ has bounded arboricity.

Our results: We focus on general CONFLICT-FREE SET COVER as well as the restricted version c -INTERSECTION SET COVER. Let us refer to the conflict-free version of c -INTERSECTION SET COVER as c -INTERSECTION CONFLICT-FREE SET COVER.

We refer to Tables 8.1 and 8.2 listing results for CONFLICT-FREE SET COVER and c -INTERSECTION CONFLICT-FREE SET COVER respectively.

- Our first result is an $f(k)|\mathcal{F}|^{o(k)}$ time lower bound for 1-INTERSECTION CONFLICT-FREE SET COVER assuming the Exponential Time Hypothesis (ETH). The lower bound holds even when $G_{\mathcal{F}}$ is restricted to bipartite graphs where INDEPENDENT SET is polynomial-time solvable. In contrast to this result, Banik et al. [14] showed hardness for their conflict-free geometric cover problem when $G_{\mathcal{F}}$ is from those classes of graphs where INDEPENDENT SET is W[1]-hard.
- For 1-INTERSECTION CONFLICT-FREE SET COVER with duplicate sets we give an $f(|\mathcal{U}|)|\mathcal{F}|^{o(|\mathcal{U}|)}$ lower bound assuming the ETH even when $G_{\mathcal{F}}$ is restricted to

$G_{\mathcal{F}}$ restriction	Solution size k	Universe size $ \mathcal{U} $
General graph	W[1]-hard even if $c = 1$ (Theorem 40)	W[1]-hard even if $c = 1$ (Lemma 42)
Empty graph	FPT, Polynomial Kernel [145]	FPT, Polynomial Kernel [145]
Bipartite	W[1]-hard even if $c = 1$ (Theorem 40)	W[1]-hard even if $c = 1$ (Lemma 42)
Chordal	FPT (Theorem 49)	FPT (Theorem 48)
Cluster	FPT (Corollary 8.3.2), Polynomial Kernel (Theorem 50)	FPT (Corollary 8.3.2) Polynomial Kernel (Theorem 50)
d -degenerate	FPT † [14]	FPT †
Nowhere Dense	FPT (Theorem 46)	FPT (Theorem 46)
Bounded #MIS	FPT (Theorem 45)	FPT (Theorem 45)

Table 8.2: Table of results: c -INTERSECTION CONFLICT-FREE SET COVER with duplicates. FPT* denotes that the problem is FPT whenever the SET COVER variant is FPT, #MIS denote number of maximal independent sets, † denotes results from existing literature (other than those cited).

bipartite graphs where the INDEPENDENT SET problem can be solved in polynomial time.

If there are no duplicate sets, the number of sets $|\mathcal{F}| \leq 2^{|\mathcal{U}|}$. Hence CONFLICT-FREE SET COVER is FPT as the trivial brute-force algorithm of choosing at most k sets from \mathcal{F} is of complexity bounded by $\binom{|\mathcal{F}|}{k} \leq \binom{|\mathcal{F}|}{|\mathcal{U}|} \leq \binom{2^{|\mathcal{U}|}}{|\mathcal{U}|} \leq 2^{|\mathcal{U}|^2}$.

- For the upper bound $\binom{|\mathcal{F}|}{|\mathcal{U}|}$, we give a matching lower bound of $2^{o(|\mathcal{U}| \log |\mathcal{F}|)}$ for any value of $|\mathcal{F}|$ as well assuming the ETH.

We note that the problem does not have a polynomial kernel as when $G_{\mathcal{F}}$ is an empty graph, the problem becomes SET COVER parameterized by universe size which does not have a polynomial kernel unless $\text{NP} \subseteq \text{coNP}/\text{poly}$ [48].

- On the positive side, we provide meta-theorems giving FPT algorithms for CONFLICT-FREE SET COVER parameterized by k whenever SET COVER is FPT and $G_{\mathcal{F}}$ belongs to graph classes which are sparse; i.e graphs of bounded degeneracy or nowhere dense graphs. This is proved using the recently introduced independence covering family [118]. Furthermore, if $G_{\mathcal{F}}$ is a dense graph like split or co-chordal, we give FPT algorithm whenever SET COVER is FPT. This algorithm works for a

large class of graphs where the number of maximal independent sets is polynomial in the number of vertices (that are sets in the family in our case).

- For c -INTERSECTION CONFLICT-FREE SET COVER, we give an FPT algorithm parameterized by k when we restrict $G_{\mathcal{F}}$ to chordal graphs. This contrasts the hardness result we have for c -INTERSECTION CONFLICT-FREE SET COVER in bipartite graphs.
- Furthermore, when we restrict $G_{\mathcal{F}}$ to a subclass of chordal graphs called cluster graphs, we obtain a polynomial kernel for c -INTERSECTION CONFLICT-FREE SET COVER parameterized by k .
- For CONFLICT-FREE SET COVER parameterized by $|\mathcal{U}|$, since solution size $k \leq |\mathcal{U}|$, the FPT results listed above for CONFLICT-FREE SET COVER parameterized by k also follow for $|\mathcal{U}|$. Furthermore, we give an FPT algorithm for CONFLICT-FREE SET COVER parameterized by $|\mathcal{U}|$ even in presence of duplicates when we restrict $G_{\mathcal{F}}$ to interval graphs via a dynamic programming algorithm using the ordering of the corresponding intervals. We extend this idea and give an FPT algorithm for chordal graphs which is a superclass of interval graphs via dynamic programming on the clique tree decomposition of the graph.
- We also study the CONFLICT-FREE SET COVER problem where there is an underlying (linearly representable) matroid on the family of subsets, and we want the solution to be an independent set in the matroid. Banik et al. [14] studied this version for a specialization of SET COVER where the sets are intervals on a real line.

We show that even the more general problem (where the sets in the family are arbitrary) is FPT when parameterized by the universe size, using the idea of dynamic programming over representative families [71].

We note that this result can be obtained as a corollary of a result by Bevern et al. [150] where they give algorithms for a generalization of our problem called

uncapacitated facility location problem with multiple matroid constraints. But our algorithm is simpler and has a better running time when the corresponding matroid has huge rank.

Structure of the Chapter: In Section 8.2, we give the hardness results of some of the variants of CONFLICT-FREE SET COVER . In Section 8.3, we give some FPT algorithms and kernels for some variants of CONFLICT-FREE SET COVER when the conflict-graph is restricted to various graph classes. Finally in Section 8.4, we give an FPT algorithm for MATROIDAL CONFLICT-FREE SET COVER parameterized by universe size.

8.2 Hardness results for Conflict-Free Set Cover

8.2.1 1-Intersection Conflict-Free Set Cover parameterized by solution size k

The problem c -INTERSECTION SET COVER is known to be in FPT [145]. On the contrary, for the conflict-free version, we show the following.

Theorem 40. 1-INTERSECTION CONFLICT-FREE SET COVER *cannot be solved in time $f(k)|\mathcal{F}|^{o(k)}$ for solution size k in bipartite graphs for any computable function f assuming the ETH.*

Proof. We give a reduction from the problem MULTICOLORED BICLIQUE [45] defined as follows:

MULTICOLORED BICLIQUE

Input: A bipartite graph $G = (A \cup B, E)$, an integer k , a partition of A into k sets A_1, A_2, \dots, A_k and a partition of B into k sets B_1, B_2, \dots, B_k .

Parameter: k

Question: Does there exist a subgraph of G isomorphic to the biclique $K_{k,k}$ with one vertex from each of the sets A_i and B_i ?

Given an instance of $(G, k, A_1, \dots, A_k, B_1, \dots, B_k)$ of MULTICOLORED BICLIQUE with $V(G) = \{v_1, v_2, \dots, v_n\}$, we construct an instance of CONFLICT-FREE SET COVER $(\mathcal{U}, \mathcal{F}, G_{\mathcal{F}}, 2k+1)$ without duplicates as follows:

We define the universe $\mathcal{U} = [2k] \cup V(G) \cup \{x\}$. Now, we associate a set corresponding to each vertex of the graph G . For a vertex v_j of $V(G)$, let S_{v_j} denote the corresponding set. For $i \in [k]$, if $v_j \in A_i$, define $S_{v_j} = \{v_j, i\}$. For $i \in [2k] \setminus [k]$, if $v_j \in B_{i-k}$, define $S_{v_j} = \{v_j, i\}$. Define a set $D = V(G) \cup \{x\}$. We have $\mathcal{F} = \bigcup_{v \in V(G)} S_v \cup \{D\}$. The graph $G_{\mathcal{F}}$ is obtained by taking the complement of the graph G , removing all the edges in $G[A]$ and $G[B]$ and adding an isolated vertex corresponding to the set D . Note that the graph $G_{\mathcal{F}}$ remains bipartite.

Note that \mathcal{F} is defined in such a way that all pairs of sets intersect in at most one element. Also, there are no duplicate sets in this instance as only the set S_{v_j} other than D contains the element v_j and only D contains the element x .

We claim that $(G, k, A_1, \dots, A_k, B_1, \dots, B_k)$ is a YES-INSTANCE of MULTICOLORED BICLIQUE if and only if $(\mathcal{U}, \mathcal{F}, G_{\mathcal{F}}, 2k+1)$ is a YES-INSTANCE of 1-INTERSECTION CONFLICT-FREE SET COVER.

Let $S = \{a_1, \dots, a_k, b_1, \dots, b_k\}$ be the vertices in G that form a multicolored biclique. Then $\mathcal{F}' = \{D, S_{a_1}, \dots, S_{a_k}, S_{b_1}, \dots, S_{b_k}\}$ covers \mathcal{U} as D covers $V(G) \cup \{x\}$ and $i \in S_{a_i}$ for $i \in [k]$ and $i \in S_{b_{i-k}}$ for $i \in [2k] \setminus [k]$. Since the edges across A and B in G are non-edges in $G_{\mathcal{F}}$ and D is an isolated vertex, \mathcal{F}' forms an independent set in $G_{\mathcal{F}}$. In the reverse direction,

let $\mathcal{F}' = \{S_1, \dots, S_{2k+1}\}$ be a solution of size $2k + 1$ covering \mathcal{U} . The set D has to be part of the solution \mathcal{F}' as only the set D contains the element x . Now note that an element $i \in [k]$ can be covered only by sets S_v where $v \in A_i$. Similarly an element $i \in [2k] \setminus [k]$ can be covered only by sets S_v where $v \in B_{i-k}$. Hence the vertices of the sets in \mathcal{F}' are such that there is at least one vertex from each of the sets A_i and B_i . Since the budget is limited to $2k$ after picking D , exactly one vertex from each of the sets A_i and B_i is contained in \mathcal{F}' . Since the vertices $\mathcal{F}' \setminus \{D\}$ form an independent set in the bipartite graph $G_{\mathcal{F}}$, the corresponding vertices form a biclique in G .

Since MULTICOLORED BICLIQUE cannot be solved in time $f(k)|V(G)|^{o(k)}$ for solution size k assuming ETH [129], the theorem follows. \square

8.2.2 Conflict-Free Set Cover parameterized by $|\mathcal{U}|$

In the section, we give lower bound results for CONFLICT-FREE SET COVER when parameterized by the universe size $|\mathcal{U}|$. We study the problem in both the cases when the family \mathcal{F} has duplicate sets and when it does not.

8.2.2.1 The family \mathcal{F} has duplicates

In this section, we study the case when duplicate sets are allowed in the family \mathcal{F} . Banik et al. [14] have the following hardness result for this case when the graph $G_{\mathcal{F}}$ is restricted to a class where finding independent set of size k is $W[1]$ -hard.

Theorem 41 ([14]). *If for a subclass of graphs \mathcal{G} , finding an independent set of size k is $W[1]$ -hard parameterized by k , then CONFLICT-FREE SET COVER parameterized by $|\mathcal{U}|$ is $W[1]$ -hard when $G_{\mathcal{F}}$ is restricted to the class \mathcal{G} .*

Bipartite graphs are one class of graphs where the INDEPENDENT SET problem can be solved in polynomial time. In contrast to Theorem 41, we show that 1-INTERSECTION

CONFLICT-FREE SET COVER on bipartite graphs is $W[1]$ -hard. Note that in Theorem 40 proven previously, the size of the universe can be much larger than the solution size k , and hence the hardness result does not follow from it.

Theorem 42. 1-INTERSECTION CONFLICT-FREE SET COVER *parameterized by $|\mathcal{U}|$ is $W[1]$ -hard on bipartite graphs.*

Proof. We again give a reduction from the $W[1]$ -hard problem MULTICOLORED BICLIQUE.

Given an instance of MULTICOLORED BICLIQUE, we construct an instance of 1-INTERSECTION CONFLICT-FREE SET COVER as follows: $\mathcal{U} = [2k]$. Let S_v denote the set corresponding to vertex v we add to \mathcal{F} . For $i \in [k]$, if $v \in A_i$, define $S_v = \{i\}$. For $i \in [2k] \setminus [k]$, if $v \in B_{i-k}$, define $S_v = \{i\}$. The graph G' is obtained by complementing the graph G and removing edges in the graphs $G[A]$ and $G[B]$. The graph G' remains bipartite. Since every set in \mathcal{F} is of size one, sets can pairwise intersect in at most one elements. Hence we can conclude that the instance we have constructed is a valid 1-INTERSECTION CONFLICT-FREE SET COVER instance.

Note that the construction is very similar to that in Theorem 40, the difference being the vertex v is not added to sets S_v .

The correctness proof follows similar to Theorem 40. □

8.2.2.2 The family \mathcal{F} has no duplicates

If there are no duplicate sets, the number of sets $|\mathcal{F}| \leq 2^{|\mathcal{U}|}$. Hence CONFLICT-FREE SET COVER is FPT as the trivial brute-force algorithm of choosing at most k sets from \mathcal{F} is of complexity bounded by $\binom{|\mathcal{F}|}{k} \leq \binom{|\mathcal{F}|}{|\mathcal{U}|} \leq \binom{2^{|\mathcal{U}|}}{|\mathcal{U}|} \leq 2^{|\mathcal{U}|^2}$. In this section, we give a lower bound of $2^{o(|\mathcal{U}| \log |\mathcal{F}|)}$ under ETH for CONFLICT-FREE SET COVER without duplicates when $G_{\mathcal{F}}$ is bipartite. We do so by giving an appropriate reduction from the following variant of MULTICOLORED BICLIQUE defined below.

SMALL MULTICOLORED BICLIQUE

Input: A bipartite graph $G = (A \cup B, E)$, an integer k , a partition of A into k sets A_1, A_2, \dots, A_k and a partition of B into k sets B_1, B_2, \dots, B_k such that $|A_i| = |B_i| = s$ where $k \leq s \leq 2^k/2k$.

Parameter: k

Question: Does there exist a subgraph of G isomorphic to the biclique $K_{k,k}$ with one vertex from each of the sets A_i and B_i ?

We first note that the reduction from 3-COLORING used in [114] can be modified so that we get the following lower bound for SMALL MULTICOLORED BICLIQUE.

Theorem 43. SMALL MULTICOLORED BICLIQUE *cannot be solved in time $2^{o(k \log s)}$ under the ETH.*

Proof. We give a reduction from 3-COLORING problem. Let G , a graph with N vertices be the instance of 3-Coloring problem.

Let $k = \frac{N \cdot \log 3}{\log s}$.

Divide vertices of G into k groups V_1, V_2, \dots, V_k of equal size, each size being $\frac{\log s}{\log 3}$.

For each set V_i , list out all the possible valid 3-colorings. There would be at most $3^{|V_i|} \leq 3^{\log s / \log 3} = 2^{\log 3 \cdot \log s / \log 3} = 2^{\log s} = s$ colorings. If there is no valid coloring for some V_i , we can conclude that we have a NO-INSTANCE of 3-COLORING. Duplicate some valid colorings so that the number of colorings is exactly s . Let us call list of colorings of V_i as P_i . Let $P = \cup_i P_i$.

Create a graph H with two copies of P , A and B as its vertex set with the corresponding partitions P_1, \dots, P_k being A_1, \dots, A_k and B_1, \dots, B_k . Let (A_i, c) and (B_i, c) denote the vertex corresponding to coloring c in sets A_i and B_i respectively. We add edges as follows:

Look at colorings $c_1 \in P_i$ and $c_2 \in P_j$. If $i \neq j$ and the colorings c_1 of $G[V_i]$ and c_2 of $G[V_j]$ together forms a valid coloring in the graph $G[V_i \cup V_j]$, add edges from vertex (A_i, c_1) to

(B_j, c_2) and from (A_j, c_2) to (B_i, c_1) .

Now we claim that $(H, A_1, \dots, A_k, B_1, \dots, B_k, k)$ is a YES-INSTANCE of SMALL MULTICOLORED BICLIQUE if and only if G has a 3-coloring. For the reverse direction, let C be a valid 3-coloring of G . Let $C|_{V_i}$ denote the coloring C restricted to V_i . We claim the vertices $(A_i, C|_{V_i})$ and $(B_i, C|_{V_i})$ forms a biclique. Suppose not. Then there is an absence of edge between two vertices $(A_{i_1}, C|_{V_{i_1}})$ and $(B_{i_2}, C|_{V_{i_2}})$. But then this means that $C|_{V_{i_1}} \cup C|_{V_{i_2}} = C|_{V_{i_1} \cup V_{i_2}}$ is not a valid coloring of $G[V_{i_1} \cup V_{i_2}]$ giving a contradiction.

For the forward direction, let the vertices $(A_1, c_1), \dots, (A_k, c_k), (B_1, c_1), \dots, (B_k, c_k)$ form a biclique. We say that $\cup_i c_i$ is a valid coloring of the graph G . Suppose not. Then there is a monochromatic edge (u, v) in G . Both u and v cannot belong to a group V_i as corresponding 3-coloring c_i is a valid 3-coloring of $G[V_i]$. So u and v belong to different groups i_1 and i_2 . But then there will not be an edge between vertices (A_{i_1}, c_{i_1}) and (B_{i_2}, c_{i_2}) as c_{i_1} and c_{i_2} together does not form a valid 3-coloring of $G[V_{i_1} \cup V_{i_2}]$. contradicting that the vertices $(A_1, c_1), \dots, (A_k, c_k), (B_1, c_1), \dots, (B_k, c_k)$ form a biclique.

Now suppose there is $2^{o(k \log s)}$ running time algorithm for SMALL MULTICOLORED BICLIQUE. Then there is a $2^{o(N \cdot \log 3)} = 2^{o(N)}$ time algorithm for 3-coloring violating the ETH. □

We give the following lower bound for CONFLICT-FREE SET COVER by giving a reduction from SMALL MULTICOLORED BICLIQUE.

Theorem 44. CONFLICT-FREE SET COVER *without duplicates* when $G_{\mathcal{F}}$ is bipartite cannot be solved in time $2^{o(|\mathcal{U}| \log |\mathcal{F}|)}$ under ETH.

Proof. Given an instance of $(G, A_1, \dots, A_k, B_1, \dots, B_k)$ of SMALL MULTICOLORED BICLIQUE with $V(G) = \{v_1, v_2, \dots, v_n\}$, we construct an instance of CONFLICT-FREE SET COVER $(\mathcal{U}, \mathcal{F}, G_{\mathcal{F}}, 2k + 1)$ without duplicates as follows:

Let us define sets $Z = \{z_1, z_2, \dots, z_{\lceil \log n \rceil}\}$ and $O = \{o_1, o_2, \dots, o_{\lceil \log n \rceil}\}$.

We define the universe $\mathcal{U} = [2k] \cup Z \cup O \cup \{x\}$.

Let us look at vertex $v_j \in V$ and construct sets $S_{v_j} \in \mathcal{F}$. Let us map j to its binary representation $b_1, b_2, \dots, b_{\lceil \log n \rceil}$ where b_i denotes the i^{th} bit of the number j . We create a set T_j as follows: for all $i \in [\lceil \log n \rceil]$, when $b_i = 0$, add z_i to T_j , else add o_i to T_j . For $i \in [k]$, if $v_j \in A_i$, define $S_{v_j} = \{i\} \cup T_j$. For $i \in [2k] \setminus [k]$, if $v_j \in B_{i-k}$, define $S_{v_j} = \{i\} \cup T_j$. Define another set $D = Z \cup O \cup \{x\}$. We have $\mathcal{F} = \bigcup_{v \in V(G)} S_v \cup \{D\}$. The graph $G_{\mathcal{F}}$ is obtained by taking the complement of the graph G , removing the edges in the graphs $G[A]$ and $G[B]$ independent and adding an isolated vertex corresponding to the set D . Note that the graph $G_{\mathcal{F}}$ remains bipartite.

Note that the construction is almost the same as in Theorem 40 but the vertices are encoded in binary form.

We now claim that $(G, k, A_1, \dots, A_k, B_1, \dots, B_k)$ is a YES-INSTANCE of SMALL MULTICOLORED BICLIQUE if and only if $(\mathcal{U}, \mathcal{F}, G_{\mathcal{F}}, 2k+1)$ is a YES-INSTANCE of CONFLICT-FREE SET COVER .

Let $S = \{a_1, \dots, a_k, b_1, \dots, b_k\}$ be the vertices in G that form a multicolored biclique. Then $\mathcal{F}' = \{D, S_{a_1}, \dots, S_{a_k}, S_{b_1}, \dots, S_{b_k}\}$ covers \mathcal{U} as D covers $Z \cup O \cup \{x\}$ and $i \in S_{a_i}$ for $i \in [k]$ and $i \in S_{b_{i-k}}$ for $i \in [2k] \setminus [k]$. Since the edges across A and B in G are non-edges in $G_{\mathcal{F}}$ and D is an isolated vertex, \mathcal{F}' forms an independent set in $G_{\mathcal{F}}$. In the reverse direction, let $\mathcal{F}' = \{S_1, \dots, S_{2k+1}\}$ be a solution of size $2k+1$ covering \mathcal{U} . The set D has to be part of the solution \mathcal{F}' as only the set D contains the element x . Now note that an element $i \in [k]$ can be covered only by sets S_v where $v \in A_i$. Similarly an element $i \in [2k] \setminus [k]$ can be covered only by sets S_v where $v \in B_{i-k}$. Hence the vertices of the sets in \mathcal{F}' are such that there is at least one vertex from each of the sets A_i and B_i . Since the budget is limited to $2k$ after picking D , exactly one vertex from each of the sets A_i and B_i is contained in \mathcal{F}' . Since the vertices $\mathcal{F}' \setminus \{D\}$ form an independent set in the bipartite graph $G_{\mathcal{F}}$, the corresponding vertices form a biclique in G .

Note that in the SMALL MULTICOLORED BICLIQUE instance, $n = 2k \cdot s \leq 2^k$. Since $\log n \leq k$, $|U| \leq 4k + 1$.

Now suppose CONFLICT-FREE SET COVER has an algorithm with running time $2^{o(|\mathcal{U}| \log |\mathcal{F}|)}$. Since $s = \frac{|\mathcal{F}|}{2k}$ and $|\mathcal{U}| \leq 4k + 1$, we have a running time of $2^{o(4k \log(2k \cdot s))} = 2^{o(k(\log s + \log k))} = 2^{o(k \log s)}$ for SMALL MULTICOLORED BICLIQUE violating the ETH. \square

8.3 Algorithms

In this section, we give algorithms for variants of CONFLICT-FREE SET COVER when the graph $G_{\mathcal{F}}$ is restricted to different graph classes.

8.3.1 Conflict-Free Set Cover parameterized by solution size k

In the following results, we restrict the graph $G_{\mathcal{F}}$.

8.3.1.1 Graphs with bounded number of maximal independent sets

Theorem 45. *When $G_{\mathcal{F}}$ is restricted to a graph where the number of maximal independent sets is polynomial in $|\mathcal{F}|$, if the restricted variant of SET COVER can be solved in $\mathcal{O}^*(f(k))$ time, then the corresponding CONFLICT-FREE SET COVER variant can be solved in $\mathcal{O}^*(f(k))$ time.*

Proof. We first note that since the maximal independent sets of a graph can be enumerated with polynomial delay (the maximum time taken between outputting two consecutive solutions) [98], they can be enumerated in time polynomial in $|\mathcal{F}|$ for the given graph $G_{\mathcal{F}}$.

For each maximal independent set I of $G_{\mathcal{F}}$, we run the $\mathcal{O}^*(f(k))$ algorithm for SET COVER

with the family \mathcal{F} containing sets corresponding to the vertices in I . Since the solution X of CONFLICT-FREE SET COVER is an independent set, $X \subseteq I'$ for some maximal independent set I' . So if the SET COVER algorithm returns YES for any I , return YES, else return NO. □

As the number of maximal independent sets in split graphs (since at most one vertex of the clique can be in the independent set), co-chordal graphs [80] and $2K_2$ -free graphs [61] is polynomial in the number of vertices and can be enumerated in polynomial time, we have the following corollary.

Corollary 8.3.1. *If SET COVER can be solved in $\mathcal{O}^*(f(k))$ time, then CONFLICT-FREE SET COVER can be solved in $\mathcal{O}^*(f(k))$ time when $G_{\mathcal{F}}$ is restricted to split graphs, co-chordal graphs or $2K_2$ -free graphs.*

8.3.1.2 Nowhere Dense graphs

Nowhere dense graph class contains several graph classes such as graphs with bounded degree, graphs with bounded local treewidth, graphs with bounded expansion and graphs that locally exclude a fixed minor.

We define the notion of k -Independence Covering Family introduced by [118].

Definition 8.3.1 (k -Independence Covering Family). *For a graph G and integer k , a family of independent sets of G is called an independence covering family for (G, k) , denoted by $\mathcal{F}(G, k)$, if for any independent set X in G of size at most k , there exists an independent set $Y \in \mathcal{F}(G, k)$ such that $X \subseteq Y$.*

In [118], the authors construct a k -independence covering family for nowhere dense graphs.

Lemma 8.3.1 ([118]). *Let G be a nowhere dense graph and k be an integer. There is a*

deterministic algorithm that runs in time

$$\mathcal{O}\left(f\left(k, \frac{1}{k}\right) \cdot n^{1+o(1)} + g(k) \cdot \binom{k^2}{k} \cdot 2^{o(k^2)} \cdot n(n+m) \log n\right)$$

and outputs a k -independence covering family for (G, k) of size $\mathcal{O}(g(k) \binom{k^2}{k} \cdot 2^{o(k^2)} \cdot n \log n)$ where f is a computable function and $g(k) = (f(k, \frac{1}{k}))^k$.

We get the following theorem.

Theorem 46. *If the restricted variant of SET COVER can be solved in $\mathcal{O}^*(h(k))$ time with solution size k and a computable function h , then the corresponding CONFLICT-FREE SET COVER variant has an algorithm with running time $\mathcal{O}^*(h(k)g(k) \binom{k^2}{k} \cdot 2^{o(k^2)})$ for nowhere dense graphs for a computable function g .*

Proof. We use Lemma 8.3.1 on $G_{\mathcal{F}}$ to get a k -independence covering family $\mathcal{F}(G_{\mathcal{F}}, k)$. For each independent set $Y \in \mathcal{F}(G_{\mathcal{F}}, k)$, we run the algorithm for SET COVER for the instance (\mathcal{U}, Y, k) in $\mathcal{O}^*(h(k))$ time. If for any of the sets Y , (\mathcal{U}, Y, k) is a YES-INSTANCE, we return YES. Otherwise we return NO.

Let X be the solution of size k . There is a set Y in $\mathcal{F}(G_{\mathcal{F}}, k)$ such that $X \subseteq Y$. Hence when we run the algorithm for SET COVER in instance (\mathcal{U}, Y, k) , since $G[Y]$ is an independent set, the algorithm will return X . \square

We note that Banik et al. [14] has proven that CONFLICT-FREE SET COVER is FPT parameterized by k if the SET COVER variant is FPT parameterized by k when $G_{\mathcal{F}}$ is a graph of bounded arboricity. The result also holds for graphs with bounded degeneracy as the degeneracy of a graph is also bounded when the arboricity is bounded. A k -Independence Covering Family can also be constructed for graphs with bounded degeneracy [118]. We note that an alternate algorithm for CONFLICT-FREE SET COVER parameterized by k when $G_{\mathcal{F}}$ has bounded degeneracy can be obtained using the ideas used for nowhere

dense graphs earlier. Note that graphs with bounded degeneracy contain many other graph classes such as planar graphs and graphs with bounded treewidth.

8.3.2 Conflict-Free Set Cover parameterized by $|\mathcal{U}|$ when \mathcal{F} has duplicates

We remind that when \mathcal{F} has no duplicates, CONFLICT-FREE SET COVER parameterized by $|\mathcal{U}|$ is trivially FPT as $|\mathcal{F}| \leq 2^{|\mathcal{U}|}$. Hence we focus on the case when there are duplicate sets in \mathcal{F} . Again we restrict the graph $G_{\mathcal{F}}$.

8.3.2.1 Interval Graphs

Before we state our result, let us focus on some properties of interval graphs. Let us order the vertices of a given interval graph G as v_1, \dots, v_n based on the increasing value of their left endpoints. Let the indices $1, \dots, n$ denote the intervals. Let $l(i)$ and $r(i)$ denote the left and right endpoints of interval i respectively.

Look at a vertex v_i and its neighborhood $N(v_i)$ in the set $\{v_{i+1}, \dots, v_n\}$. Let $v_j, v_k \in \{v_{i+1}, \dots, v_n\}$ such that $(v_i, v_j) \in E(G)$ and $(v_i, v_k) \notin E(G)$. By definition, v_i and v_j has an edge if intervals i and j intersect. Hence $l(j) \leq r(i)$. Also since intervals i and k do not intersect, $r(i) \leq l(k)$. Hence we have $l(j) \leq l(k)$. Since this is true for any non-neighbor of v_i in $\{v_{i+1}, \dots, v_n\}$, we have shown that all the non-neighbors of v_i to its right comes after the last neighbor of v_i to its right. We make use of this ordering to give a dynamic programming algorithm for CONFLICT-FREE SET COVER with duplicates on interval graphs. Note that the ordering can be obtained in time linear in $|V(G)|$ by arranging them according to their leftmost endpoints.

Theorem 47. CONFLICT-FREE SET COVER *with duplicate sets when $G_{\mathcal{F}}$ is restricted to interval graphs can be solved in $\mathcal{O}^*(2^{|\mathcal{U}|})$ time.*

Proof. Let the sets of $\mathcal{F} = \{S_1, \dots, S_m\}$ be ordered in the reverse order of the ordering described above. For each subset $W \subseteq \mathcal{U}$, and $i \in [m]$, define $DP[W, i]$ as the size of the minimum set $X \subseteq \{S_1, \dots, S_i\}$ such that X covers W and vertices of X are independent in $G_{\mathcal{F}}$. Initially, set $DP[\emptyset, 0] = 0$ and $DP[X, 0] = \infty$ when $X \neq \emptyset$. We have the following recursive formula for $DP[W, i]$.

$$DP[W, i] = \min \{ 1 + DP[W \setminus S_i, \ell], DP[W, i - 1] \}$$

where ℓ is the index of the rightmost non-neighbor of S_i in $G_{\mathcal{F}}[\{S_1, \dots, S_{i-1}\}]$.

The correctness proof of the above equation is as follows.

Let X be the optimal solution for $DP[W, i]$. The subfamily X either contains the set S_i or it does not. When X does not contain S_i , then it is a valid candidate for $DP[W, i - 1]$ and hence $|X| \geq DP[W, i - 1]$. When it contains S_i , $X \setminus \{S_i\}$ is a valid candidate for $DP[W \setminus S_i, \ell]$ and hence $|X| - 1 \geq DP[W \setminus S_i, \ell]$. Hence $DP[W, i] \geq \min \{ 1 + DP[W \setminus S_i, \ell], DP[W, i - 1] \}$.

Let Y be the optimal solution for $DP[W \setminus S_i, \ell]$. Then $Y \cup S_i$ is a valid candidate for $DP[W, i]$ since $\{S_1, \dots, S_{\ell}\}$ contains only non-neighbors of S_i as all the neighbors of S_i follows after the rightmost non-neighbor of S_i which is S_{ℓ} . Hence $DP[W, i] \leq 1 + DP[W \setminus S_i, \ell]$. Let Z be the optimal solution for $DP[W, i - 1]$. Then Z is also a valid candidate for $DP[W, i]$. Hence $DP[W, i] \leq DP[W, i - 1]$.

The entry $DP[W, m]$ contains the size of the minimum-sized solution of CONFLICT-FREE SET COVER. The number of subproblems is $\sum_{j \in [\mathcal{U}]} \binom{|\mathcal{U}|}{j} \cdot m$ and at each subproblem $\mathcal{O}(m)$ time is spent to find ℓ . Hence the running time is $\sum_{j \in [\mathcal{U}]} \binom{|\mathcal{U}|}{j} \cdot \mathcal{O}(m) = \mathcal{O}^*(2^{|\mathcal{U}|})$. \square

Now we give a $\mathcal{O}^*(3^{|\mathcal{U}|})$ -time dynamic programming algorithm for chordal graphs which is a superclass of interval graphs.

8.3.2.2 Chordal Graphs

A *clique tree decomposition* is a tree decomposition T where for all nodes $i \in V(T)$, the vertices of in the bag X_i are such that $G[X_i]$ forms a clique. All chordal graphs have clique tree decompositions that can be found in polynomial time [80]. Given a clique tree decomposition, it can be converted to a nice clique tree decomposition in polynomial time using Lemma 2.3.1. Note from Lemma 2.3.1 that every bag of the new nice tree decomposition is a subset of some bag of the original tree decomposition. Hence every bags in the nice tree decompositions are also cliques.

In the theorem below, we give an algorithm for CONFLICT-FREE SET COVER with duplicates on chordal graphs using dynamic programming on the nice clique tree decomposition of the graph.

Theorem 48. CONFLICT-FREE SET COVER *with duplicates on chordal graphs can be solved in $O^*(3^{|\mathcal{U}|})$ running time.*

Proof. For the instance $(\mathcal{U}, \mathcal{F}, G_{\mathcal{F}}, k)$ of CONFLICT-FREE SET COVER , let T be the tree of the nice clique tree decomposition of the chordal graph $G_{\mathcal{F}}$. For a node $i \in V(T)$, let T_i denote the subtree rooted at node i , V_i denote the vertices of G in the bags of nodes of T_i and X_i denote the vertices in the bag of node i . Note that since we are looking for a solution that is also independent set in the chordal graph, from each bag no more than one vertex can be in the solution as $G[X_i]$ forms a clique.

For each subset $W \subseteq \mathcal{U}$, node $i \in V(T)$ and $x \in X_i$, let $DP[W, i, x]$ denote the size of the minimum-sized independent set Y of the graph $G[V_i]$ covering W such that $x \in Y$. Node x can take empty value \emptyset as well to denote no vertex is picked from the bag X_i . Initially, set all entries to ∞ denoting that no such solution exists. We have the following recurrence relations for each type of node in T to compute $DP[W, i, x]$:

- Leaf Node:

$$DP[W, i, \emptyset] = \begin{cases} 0 & \text{if } W = \emptyset, \\ \infty & \text{otherwise} \end{cases}$$

- Introduce Node: Let i be the parent of node j and vertex v is introduced in X_i .

$$DP[W, i, x] = \begin{cases} DP[W, j, x] & \text{if } x \neq v \\ 1 + DP[W \setminus S_v, j, \emptyset] & \text{when } x = v \end{cases}$$

- Forget Node: Let i be the parent of node j and vertex v is forgotten in X_i .

$$DP[W, i, x] = \begin{cases} DP[W, j, x] & \text{if } x \neq \emptyset \\ \min \{DP[W, j, \emptyset], DP[W, j, v]\} & \text{when } x = \emptyset \end{cases}$$

- Join Node: Let i be the parent of two nodes j and j' and $X_i = X_j = X_{j'}$.

$$DP[W, i, x] = \begin{cases} \min_{W_1 \subseteq W} \{DP[W_1, j, x] + DP[W \setminus W_1, j', x] - 1\} & \text{if } x \neq \emptyset \\ \min_{W_1 \subseteq W} \{DP[W_1, j, \emptyset] + DP[W \setminus W_1, j', \emptyset]\} & \text{when } x = \emptyset \end{cases}$$

The entry $DP[\mathcal{U}, r, \emptyset]$ contains the size of the minimum-sized solution of CONFLICT-FREE SET COVER where r is the root of the tree. The number of subproblems is $\mathcal{O}(\sum_{j=1}^{|\mathcal{U}|} \binom{|\mathcal{U}|}{j} \cdot |T| \cdot |\mathcal{F}|)$. The maximum time spent on computing $DP[W, i, x]$ where $|W| = j$ is $\mathcal{O}(2^j)$ for going over all subsets W_1 at the join node. Hence the overall running time is $\mathcal{O}^*(\sum_{j=1}^{|\mathcal{U}|} \binom{|\mathcal{U}|}{j} \cdot 2^j) = \mathcal{O}^*(3^{|\mathcal{U}|})$.

Correctness of Recurrence Relations:

For ease of writing, let us denote the terms present in the left hand side of the equation as *LHS* and on the right hand side of the equation as *RHS*. For each recurrence relation

defined above, we prove its correctness by showing inequality in both sides. We use the term *optimal* solution for a DP entry to denote the minimum-sized conflict-free set cover corresponding to the entry and *candidate* solution for a DP entry to denote any conflict-free set cover corresponding to the entry (need not be of minimum size).

- Introduce Node:

Let X be the optimal solution for the entry $DP[W, i, x]$. By definition $x \in X$. If $x \neq v$, X is also a candidate solution for $DP[W, j, x]$ as $V_i \setminus \{v\} = V_j$. If $x = v$, $X \setminus \{v\}$ is a candidate solution for $DP[W, j, \emptyset]$ as no $y \in X_i, y \neq v$ can be in $X \setminus \{v\}$ since $G[X_i]$ is a clique. In either case, the value at *RHS* can be either the size of *LHS* or even lower. Hence, $LHS \geq RHS$.

Let Y be the optimal solution for $DP[W, j, x]$. If $x \neq \emptyset$, the set Y is also a candidate solution for $DP[W, i, x]$. Hence $LHS \leq RHS$. If $x = \emptyset$, look at Z , the solution for $DP[W \setminus S_v, j, x]$. Since all the edges of v in the graph $G[V_i]$ is in bag X_i , $Z \cup \{v\}$ is also an independent set and it covers W . Hence both Y and Z are candidate solutions for $DP[W, i, x]$ when $x = \emptyset$. Hence $LHS \leq RHS$.

- Forget Node:

Let X be the optimal solution for $DP[W, i, x]$ such that $X \cap X_i = \{x\}$ with $x \neq \emptyset$. Since $V_j = V_i$, X is also a solution for $G[V_j]$ such that $X \cap X_i = \{x\}$ and hence a candidate solution for $DP[W, j, x]$. Hence $LHS \geq RHS$. Similarly we can prove the inequality in the other direction.

When $x = \emptyset$, let X be the optimal solution for $DP[W, i, x]$ such that $X \cap X_i = \emptyset$. Since $V_j = V_i$, X is also a solution for $G[V_j]$ such that $X \cap X_i = \emptyset$ and hence a candidate solution for $DP[W, j, x]$. Also if $v \in X$, X is a candidate solution for $DP[W, j, v]$ as well. If $v \in X$, $DP[W, j, \emptyset] \geq DP[W, j, v]$. Hence $LHS \geq RHS$.

Let Y be the optimal solution for the minimum of two entries $DP[W, j, \emptyset]$ and

$DP[W, j, v]$. If the minimum is $DP[W, j, v]$, then $Y \setminus \{v\}$ is a candidate solution of $DP[W, i, \emptyset]$. Else Y is also a candidate solution of $DP[W, i, \emptyset]$. Hence $LHS \leq RHS$.

- Join Node:

Let X be the optimal solution for the entry $DP[W, i, x]$. When $x \neq \emptyset$ and $G[X_i]$ forms a clique, $X \cap X_i = \{x\}$. Let W_1 and W_2 be the subset of elements covered by $Y_j = X \cap V_j$ and $Y_{j'} = X \cap V_{j'}$ respectively. Note that since X covers W , $W_1 \cup W_2 = W$. Since X is an independent set, Y_j and $Y_{j'}$ are independent sets as well as they both are subsets of X . Note that $Y_j \cap Y_{j'} = \{x\}$. Hence Y_j and $Y_{j'}$ respectively are candidate solutions to entries $DP[W_1, j, x]$ and $DP[W \setminus W_1, j', x]$ as $W \setminus W_1 \subseteq W_2$. Since x is the only entry common to both of them, we have $LHS \geq RHS$.

Let Z_j and $Z_{j'}$ be the optimal solutions for the entries $DP[W_1, j, x]$ and $DP[W_2, j', x]$ where $W_2 = W \setminus W_1$. Since $X_i = X_j = X_{j'}$ and $G[X_i]$ forms a clique, $Z_j \cap X_j = Z_{j'} \cap X_{j'} = \{x\}$. Look at the set $Z = Z_j \cup Z_{j'}$. The set Z is an independent set since Z_j and $Z_{j'}$ are independent sets and since there are no edges across $G[V_j \setminus X_i]$ and $G[V_{j'} \setminus X_i]$ by the definition of tree decomposition. Hence Z is a candidate solution for the entry $DP[W, i, x]$ of size $|Z_j| + |Z_{j'}| - 1$. Therefore $LHS \leq RHS$.

When $x = \emptyset$, using similar arguments we can prove that $LHS = RHS$.

□

8.3.3 c -Intersection Conflict-Free Set Cover parameterized by k

8.3.3.1 FPT algorithm for Chordal Graphs

When $G_{\mathcal{F}}$ is a chordal graph, we could not come up with a meta-theorem like we had earlier in Section 8.3.1 for split graphs, nowhere dense graphs etc which gave an FPT algorithm for CONFLICT-FREE SET COVER given that the restricted version of SET COVER has

an FPT algorithm. Hence we focus on a particular restriction of SET COVER known to be FPT which is c -INTERSECTION SET COVER and give an FPT algorithm for the conflict-free version. Note that on the contrary, Theorem 40 shows that the problem is $W[1]$ -hard when $G_{\mathcal{F}}$ is bipartite even when $c = 1$.

Given the instance $(\mathcal{U}, \mathcal{F}, G_{\mathcal{F}}, k)$. We start the algorithm with the following reduction rule.

Reduction Rule 6. *If there is a set $S \in \mathcal{F}$ such that $|S| > ck$, then put S in the solution and drop k by 1. The new instance is $(\mathcal{U}', \mathcal{F}', G'_{\mathcal{F}}, k - 1)$ where $\mathcal{U}' = \mathcal{U} \setminus S$, $\mathcal{F}' = \mathcal{F} \setminus N[S]$ and $G'_{\mathcal{F}} = G_{\mathcal{F}}[\mathcal{F}']$.*

Claim 8.3.1. *Reduction Rule 6 is safe.*

Proof. Let $I' = (\mathcal{U}', \mathcal{F}', G'_{\mathcal{F}}, k')$ be the instance of c -INTERSECTION CONFLICT-FREE SET COVER after applying Reduction Rule 6 to instance $I = (\mathcal{U}, \mathcal{F}, G_{\mathcal{F}}, k)$ for a set $S \in \mathcal{F}$. We show that I is a YES-INSTANCE if and only if I' is a YES-INSTANCE.

Let $\mathcal{X} \subseteq \mathcal{F}$ be a solution of size at most k . We claim that $S \in \mathcal{X}$. Suppose not. The elements of S has to be covered by the other sets in \mathcal{F} . We know that for any set $S' \in \mathcal{F}$, $|S' \cap S| \leq c$. Since $|\mathcal{X}| \leq k$, \mathcal{X} can cover only at most ck elements of S . Since $|S| > ck$, \mathcal{X} do cover the set S giving a contradiction.

We claim that the set $\mathcal{X}' = \mathcal{X} \setminus S$ is a solution of size at most $k' \leq k - 1$ to the instance I' . Suppose not. Note that all the sets in \mathcal{X}' are present in $\mathcal{F}' = \mathcal{F} \setminus N[S]$ as they cannot be present in $N[S]$ which would contradict the fact that \mathcal{X} is an independent set in $G_{\mathcal{F}}$. Since \mathcal{X} covers \mathcal{U} , \mathcal{X}' covers $\mathcal{U}' = \mathcal{U} \setminus S$. Also since \mathcal{X} is an independent set in $G_{\mathcal{F}}$, \mathcal{X}' is an independent set in $G'_{\mathcal{F}}$. Hence I' is a YES-INSTANCE.

Conversely, let Y' be a solution of size k' to the instance I' . We claim that $Y = Y' \cup S$ is a solution of size at most k to the instance I . Since Y' is an independent set in $G'_{\mathcal{F}}$ and $\mathcal{F}' = \mathcal{F} \setminus N[S]$, Y is an independent set in $G_{\mathcal{F}}$. Also since $\mathcal{U}' = \mathcal{U} \setminus S$, Y covers \mathcal{U} . \square

We apply Reduction Rule 6 exhaustively. Note that by applying Reduction Rule 6, we

introduce duplicate sets. Afterwards, we can assume that the size of every set in \mathcal{F} has size at most ck . We now apply the following reduction rule.

Reduction Rule 7. *If $|\mathcal{U}| > ck^2$, return NO.*

Since every set in \mathcal{F} has size at most ck , a solution of size at most k can cover at most ck^2 elements. Hence if $|\mathcal{U}| > ck^2$, there is no solution of size k and hence we return NO.

After applying reduction rules 6 and 7 exhaustively in order, we get an instance where the universe size $|\mathcal{U}| \leq ck^2$, a function of k . Hence the problem can now be treated as an instance of CONFLICT-FREE SET COVER parameterized by $|\mathcal{U}|$. We use Theorem 48 to get an FPT algorithm with running time $\mathcal{O}^*(3^{|\mathcal{U}|}) = \mathcal{O}^*(3^{ck^2})$. Hence we have the following theorem.

Theorem 49. *c -INTERSECTION CONFLICT-FREE SET COVER when $G_{\mathcal{F}}$ is a chordal graph has an algorithm with a running time of $\mathcal{O}^*(3^{ck^2})$.*

8.3.3.2 Polynomial Kernel in cluster graphs

In this section, we show a polynomial kernel for cluster graphs which is a subclass of chordal graphs.

We initially apply reduction rules 6 and 7 exhaustively in order. Hence we can assume that the universe size is at most ck^2 .

We first claim that there are only $\binom{ck^2}{c+1}$ distinct sets present in \mathcal{F} . Let us look at an arbitrary subset A of $c+1$ elements from \mathcal{U} . There is only one set $S \in \mathcal{F}$ such that $A \subseteq S$. Suppose there also exist $S' \in \mathcal{F}, S' \neq S$ such that $A \subseteq S'$. Then we have $A \subseteq S \cap S'$. Hence $|S' \cap S| > c$ giving a contradiction.

Hence we can create an injective map from each distinct set in \mathcal{F} of size at least $c+1$ to a subset of $c+1$ elements of \mathcal{U} . Since there are at most $\binom{|\mathcal{U}|}{c+1}$ such subsets, there are at most $\binom{ck^2}{c+1}$ distinct elements in \mathcal{F} of size at least $c+1$. The number of distinct sets in \mathcal{F} of size

at most c is also bounded by $\binom{|\mathcal{U}|}{c+1} \leq \binom{ck^2}{c+1}$. Hence we can conclude that the total number of distinct elements in \mathcal{F} is at most $\binom{ck^2}{c+1}$.

Hence to bound the size of \mathcal{F} , we only need to bound the number of duplicates in \mathcal{F} .

Let C_1, C_2, \dots, C_p the components in the cluster graph $G_{\mathcal{F}}$, each component being a clique. We have the following reduction rule.

Reduction Rule 8. *If a component C_i where $i \in [p]$ has two vertices v and v' where the set corresponding to both vertices is the same set S , delete v' from \mathcal{F} .*

Both the vertices v and v' cover the same set S and have the same closed neighborhood set which is the entire clique. Since a solution will contain only at most one vertex from C_i as it is a clique, the reduction rule 8 is safe.

We apply reduction rule 8 to all components C_i for $i \in [p]$. Since all the sets in C_i are distinct afterwards, we have $|C_i| \leq \binom{ck^2}{c+1}$.

We have the following reduction rule to take care of duplicate sets among different components C_i .

Reduction Rule 9. *For each distinct set $S \in \mathcal{F}$, if there are more than k vertices whose corresponding set is S , keep arbitrarily selected $k+1$ vertices whose set is S and delete the rest of the vertices.*

After applying this rule, we can conclude that every set in \mathcal{F} has at most k duplicates.

Claim 8.3.2. *Reduction Rule 9 is safe on instances of c -INTERSECTION CONFLICT-FREE SET COVER where $G_{\mathcal{F}}$ is a cluster graph.*

Proof. Let $I' = (\mathcal{U}', \mathcal{F}', G'_{\mathcal{F}}, k')$ be the instance of c -INTERSECTION CONFLICT-FREE SET COVER after applying Reduction Rule 9 to instance $I = (\mathcal{U}, \mathcal{F}, G_{\mathcal{F}}, k)$ for a set $S \in \mathcal{F}$. We show that I is a YES-INSTANCE if and only if I' is a YES-INSTANCE.

Let X be a solution of size at most k for I . We construct a subset of vertices X' in the instance I' as follows. The set X' is initially empty. In phase 1, for each vertex $v \in X$, if the number of duplicates of the corresponding set S_v is not more than k in I , then v is also present in I' . Add v to X' . Mark v and the corresponding component containing v . If the number of duplicates of S_v is more than k in I , we do nothing.

After we do this for every vertex in X , phase 2 begins. Every unmarked vertex $v \in X$ has more than k duplicates in I . For each such vertex v we add a vertex w to X' from an unmarked component whose corresponding set is S_v . Mark the corresponding component containing w .

Note that the procedure to construct X' terminates without fail. This is because there is an unmarked component containing vertex S_v at every step where a vertex is added since we keep $k + 1$ duplicates for v which is present in different components of $G_{\mathcal{F}}$.

We claim that X' is a solution for the instance I' . Clearly, X' covers \mathcal{U} as the sets corresponding to each vertex in X' remain the same as X . Since at each time, a vertex in X' is added from an unmarked cluster, X' also forms an independent set.

Conversely, a solution Y for I' is also a solution for I as all the vertices of Y are also present in I . □

After applying reduction rules 1 to 4 exhaustively in order, it is easy to see that we get a kernel for c -INTERSECTION CONFLICT-FREE SET COVER with universe size ck^2 and family size $(k + 1) \cdot \binom{ck^2}{c+1}$. We have the following theorem.

Theorem 50. *c -INTERSECTION CONFLICT-FREE SET COVER parameterized by k when $G_{\mathcal{F}}$ is a cluster graph has a kernel with universe size ck^2 and family size $(k + 1) \cdot \binom{ck^2}{c+1}$.*

Using the kernel, we get a better FPT algorithm when $G_{\mathcal{F}}$ is a cluster graph by going over all the k -sized subsets of \mathcal{F} .

Corollary 8.3.2. *c -INTERSECTION CONFLICT-FREE SET COVER parameterized by k*

when $G_{\mathcal{F}}$ is a cluster graph has an FPT algorithm with running time $\mathcal{O}^*((k+1) \cdot \binom{ck^2}{c+1})^k) = \mathcal{O}^*(k^{\mathcal{O}(ck)})$.

8.4 Matroidal Conflict-free Set Cover

In this section, we study the MATROIDAL CONFLICT-FREE SET COVER problem where the conflicting condition is being an independent set in a (representable) matroid.

Let \mathbb{F}_{p^ℓ} denote a finite field of order p^ℓ where p is a prime and ℓ is a positive integer. Also, we denote by \mathbb{Q} the field of rationals. Let us first define the MATROIDAL CONFLICT-FREE SET COVER problem as follows.

MATROIDAL CONFLICT-FREE SET COVER

Input: A universe \mathcal{U} , a family \mathcal{F} of subsets of \mathcal{U} , a linear representation of a matroid $M = (\mathcal{F}, \mathcal{I})$ over a field \mathbb{F} where $\mathbb{F} = \mathbb{F}_{p^\ell}$ or \mathbb{F} is \mathbb{Q} and an integer k .

Question: Is there a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ of size at most k such that $\bigcup_{F \in \mathcal{F}'} F = \mathcal{U}$ and \mathcal{F}' forms an independent set in M ?

Note that we need the linear representation of the matroid M over a field \mathbb{F} where $\mathbb{F} = \mathbb{F}_{p^\ell}$ or \mathbb{F} is \mathbb{Q} . This is due to technical reasons which will be revealed later.

We give a dynamic programming algorithm for MATROIDAL CONFLICT-FREE SET COVER containing duplicate sets using computation of representative sets noting that the similar ideas used in [14] for INTERVAL COVERING can be extended to MATROIDAL CONFLICT-FREE SET COVER .

For $W \subseteq \mathcal{U}$, let \mathcal{B}^W denote the collection of subfamilies X of \mathcal{F} of size at most k such that X covers W and forms an independent set in the matroid M .

$$\mathcal{B}^W = \{X \subseteq \mathcal{F} \mid |X| \leq k, W \subseteq \bigcup_{S \in X} S \text{ and } X \in \mathcal{I}\}$$

Note that $\mathcal{B}^{\mathcal{U}}$ contains all the solutions of size at most k of MATROIDAL CONFLICT-FREE SET COVER . Hence we solve the MATROIDAL CONFLICT-FREE SET COVER problem by checking whether $\mathcal{B}^{\mathcal{U}}$ is empty or not.

Definition 8.4.1 (q -representative family [127]). *Let $M = (E, \mathcal{I})$ be a matroid and \mathcal{A} be a family of sets of size p in M . For sets $A, B \subseteq E$, we say that A fits B if $A \cap B = \emptyset$ and $A \cup B \in \mathcal{I}$. A subfamily $\hat{\mathcal{A}} \subseteq \mathcal{A}$ is said to q -represent \mathcal{A} if for every set B of size q such that there is an $A \in \mathcal{A}$ that fits B , there is an $\hat{A} \in \hat{\mathcal{A}}$ that also fits B . We use $\hat{\mathcal{A}} \subseteq_{rep}^q \mathcal{A}$ to denote that $\hat{\mathcal{A}}$ q -represents \mathcal{A} .*

Lemma 8.4.1 ([71]). *For a matroid $M = (E, \mathcal{I})$ and $S \subseteq E$, if $\mathcal{S}_1 \subseteq_{rep}^q \mathcal{S}$ and $\mathcal{S}_2 \subseteq_{rep}^q \mathcal{S}_1$, then $\mathcal{S}_2 \subseteq_{rep}^q \mathcal{S}$.*

Note that $\mathcal{B}^{\mathcal{U}}$ is nonempty if and only if $\hat{\mathcal{B}}^{\mathcal{U}} \subseteq_{rep}^0 \mathcal{B}^{\mathcal{U}}$ is nonempty. Let us define \mathcal{B}^{Wj} as the subset of \mathcal{B}^W containing sets of size exactly j . We use $\hat{\mathcal{B}}^W \subseteq_{rep}^{1, \dots, k} \mathcal{B}^W$ to denote that $\hat{\mathcal{B}}^W$ contains the union of all the i -representative families of \mathcal{B}^W where $1 \leq i \leq k$. In other words,

$$\hat{\mathcal{B}}^W = \bigcup_{j=1}^k (\hat{\mathcal{B}}^{Wj} \subseteq_{rep}^{k-j} \mathcal{B}^{Wj})$$

Lemma 8.4.2 ([115]). *Let $M = (E, \mathcal{I})$ be a linear matroid of rank n and \mathcal{S} be a family of t independent sets of size p . Let A be a $n \times |E|$ matrix representation of M over a field \mathbb{F} where $\mathbb{F} = \mathbb{F}_{p^\ell}$ or \mathbb{F} is \mathbb{Q} . Then there is a deterministic algorithm to compute $\hat{\mathcal{S}} \subseteq_{rep}^q \mathcal{S}$ of size $np \binom{p+q}{p}$ in $\mathcal{O}(\binom{p+q}{p} t p^3 n^2 + t \binom{p+q}{p}^{\omega-1} (pn)^{\omega-1}) + (n + |E|)^{O(1)}$ operations over \mathbb{F} where ω is the matrix multiplication exponent.*

Note that Lemma 8.4.2 is applicable only when the matroid is represented over a field \mathbb{F} where $\mathbb{F} = \mathbb{F}_{p^\ell}$ or \mathbb{F} is \mathbb{Q} . This is why we imposed a similar restriction for the matroid representation in the definition of MATROIDAL CONFLICT-FREE SET COVER .

Theorem 51. MATROIDAL CONFLICT-FREE SET COVER can be solved in $\mathcal{O}^*(2^{(\omega+1) \cdot |\mathcal{U}|})$ time where ω is the matrix multiplication exponent.

Proof. Let \mathcal{D} be an array of size $2^{|\mathcal{U}|}$ with $\mathcal{D}[W]$ storing the family $\hat{\mathcal{B}}^W \subseteq_{rep}^{1,\dots,k} \mathcal{B}^W$. We compute the entries of \mathcal{D} in the increasing order of subsets of \mathcal{U} . To do so we compute the following:

$$(8.1) \quad \mathcal{N}^W = \bigcup_{S_i \in \mathcal{F}} (\mathcal{D}[W \setminus S_i] \bullet S_i) \cap \mathcal{I}$$

where $\mathcal{A} \bullet \mathcal{B} = \{A \cup B \mid A \in \mathcal{A} \text{ and } B \in \mathcal{B} \text{ and } A \cap B = \emptyset\}$.

We show that $\mathcal{N}^W \subseteq_{rep}^{1,\dots,k} \mathcal{B}^W$. Let $S \in \mathcal{B}^{Wj}$ and Y be a set of size $k - j$ such that $S \cap Y = \emptyset$ and $S \cup Y \in \mathcal{I}$. We give a set $\hat{S} \in \mathcal{N}^{Wj}$ such that $\hat{S} \cap Y = \emptyset$ and $\hat{S} \cup Y \in \mathcal{I}$.

Let $S = \{S_1, S_2, \dots, S_j\}$. Let $S' = S \setminus \{S_j\}$. Let $Y' = Y \cup \{S_j\}$. Then, $|S'| = j - 1$ and $|Y'| = k - j + 1$. Since S' covers $W \setminus S_j$, $S' \in \mathcal{B}^{(W \setminus S_j)(j-1)}$. By definition, $\mathcal{D}[W \setminus S_j]$ contains $\hat{\mathcal{B}}^{(W \setminus S_j)(j-1)} \subseteq_{rep}^{k-j+1} \mathcal{B}^{(W \setminus S_j)(j-1)}$ and hence a set $S^* \in \mathcal{D}[W \setminus S_j]$ such that $S^* \cap Y' = \emptyset$ and $S^* \cup Y' \in \mathcal{I}$. From equation (8.1), $S^* \cup \{S_j\} \in \mathcal{N}^W$. The set $\hat{S} = S^* \cup \{S_j\}$ is such that $\hat{S} \cap Y = \emptyset$ and $\hat{S} \cup Y \in \mathcal{I}$. Hence $\mathcal{N}^W \subseteq_{rep}^{1,\dots,k} \mathcal{B}^W$.

We store $\hat{\mathcal{N}}^W \subseteq_{rep}^{1,\dots,k} \mathcal{N}^W$ in $\mathcal{D}[W]$. The sets $\hat{\mathcal{N}}^{Wj}$ are computed using Lemma 8.4.2. We have $\hat{\mathcal{N}}^{Wj} \subseteq_{rep}^{k-j} \mathcal{N}^{Wj} \subseteq_{rep}^{k-j} \mathcal{B}^{Wj}$ for all $1 \leq j \leq k$. Hence from Lemma 8.4.1, we have $\mathcal{D}[W] = \hat{\mathcal{N}}^W \subseteq_{rep}^{1,\dots,k} \mathcal{B}^W$.

We now focus on the running time to compute $\mathcal{D}[W]$ and the size of $\mathcal{D}[W]$. Assume that $\mathcal{D}[Y]$ is precomputed for all subsets $Y \subseteq W$. We have $|\mathcal{D}[Y]| = |\hat{\mathcal{N}}^Y| = \sum_{j=1}^k |\hat{\mathcal{N}}^{Yj}|$. From Lemma 8.4.2, $|\hat{\mathcal{N}}^{Yj}| \leq |\mathcal{F}| \cdot k \cdot \binom{k}{j}$. Hence from equation (8.1), putting $Y = W \setminus S_i$, we have $|\mathcal{N}^{Wj}| \leq |\mathcal{F}|^2 \cdot k \cdot \binom{k}{j}$. Using Lemma 8.4.2, the time to compute $\hat{\mathcal{N}}^{Wj} \subseteq_{rep}^{k-j} \mathcal{N}^{Wj}$ is $\mathcal{O}^* \left(\binom{k}{j}^2 + \binom{k}{j}^\omega \right)$ where ω is the exponent for matrix multiplication. Hence the total time to compute $\mathcal{D}[W]$ is $\sum_{j=1}^k \mathcal{O}^* \left(\binom{k}{j}^\omega \right) = \mathcal{O}^*(2^{\omega k})$. The size of $\mathcal{D}[W]$ is $\mathcal{O}(|\mathcal{F}| \cdot k \cdot \sum_{j=1}^k \binom{k}{j}) = \mathcal{O}(2^k \cdot k \cdot |\mathcal{F}|)$.

The overall running time to check if $\mathcal{D}[U]$ is empty or not is bounded by $\mathcal{O}^*(2^{|\mathcal{U}|} \cdot 2^{\omega k}) = \mathcal{O}^*(2^{\omega|\mathcal{U}|+|\mathcal{U}|}) = \mathcal{O}^*(10.361^{|\mathcal{U}|})$. \square

We note that an FPT algorithm for MATROIDAL CONFLICT-FREE SET COVER parameterized by k can be obtained as a corollary of a result by Bevern et al. [150]. The authors give an algorithm for a generalization of SET COVER called uncapacitated facility location problem with multiple matroid constraints. This algorithm also uses the idea of representative families that we use. But the algorithm involves further sophistication as they work on a general problem. The running time for MATROIDAL CONFLICT-FREE SET COVER from Bevern et al is $2^{O(r \log r)} n^2$ where r is the rank of the matroid. The rank r is bounded by the universe size $|\mathcal{U}|$. The running time is $2^{O(|\mathcal{U}| \log |\mathcal{U}|)} n^2$ when $r = \mathcal{O}(|\mathcal{U}|)$ in which case our algorithm from Theorem 51 with running time $\mathcal{O}^*(2^{(\omega+1) \cdot |\mathcal{U}|})$ is better. But when the rank is smaller, the algorithm by Bevern et al. is better.

8.5 Conclusion

We have initiated a systematic study of CONFLICT-FREE SET COVER with various parameterizations and restrictions to $G_{\mathcal{F}}$. When parameterized by the solution size k and when the restricted SET COVER variant is FPT parameterized by k , we have shown $W[1]$ -hardness for the corresponding CONFLICT-FREE SET COVER variant when the conflict graph $G_{\mathcal{F}}$ is bipartite and gave FPT algorithms when $G_{\mathcal{F}}$ is nowhere dense or has bounded number of independent sets. When parameterized by the universe size (hence SET COVER variant is FPT), we have shown $W[1]$ -hardness when $G_{\mathcal{F}}$ is bipartite and gave FPT algorithms when $G_{\mathcal{F}}$ is chordal, nowhere dense or has bounded number of independent sets. One open question is to identify a general characterization for the graph classes of $G_{\mathcal{F}}$ when CONFLICT-FREE SET COVER becomes FPT for the above two cases.

We gave an FPT algorithm for c -INTERSECTION CONFLICT-FREE SET COVER when $G_{\mathcal{F}}$ is a chordal graph but only managed to find a polynomial kernel for cluster graphs, a subgraph of chordal graphs. Finding a polynomial kernel for c -INTERSECTION CONFLICT-FREE SET COVER when $G_{\mathcal{F}}$ is a chordal graph remains open.

Chapter 9

Fair Vertex Deletion problems

9.1 Introduction

Recently, there is an interest in vertex deletion problems where along with optimizing the vertex deletion set, we want the deletion set to be fair in the sense that it does not have too many vertices from the neighborhood of any vertex [130, 105, 104]. It can be viewed as spreading the cost of the deletion set to all vertices such that the cost is not too high for anyone.

Let us formalize the problem by introducing the following notions. Given a graph $G = (V, E)$ and a positive integer d , a set $S \subseteq V$ is *fair* if it contains at most d vertices from the neighborhood of each vertex. That is, for each vertex $v \in G$, $|N(v) \cap S| \leq d$. We call d the *fairness factor* of S .

Given a set $S \subseteq V(G)$, checking whether S is a fair set can be done in polynomial time by going over the neighborhoods of all the vertices in G and counting the vertices in S in it. If for some vertex v , $|N(v) \cap S| > d$, we say that the fairness constraint for v with respect to S is violated.

A graph property Π is the same as a graph class which is a collection of graphs. We define

Π -FAIR VERTEX DELETION problem as follows.

Π -FAIR VERTEX DELETION

Input: A graph $G = (V, E)$ and $k, d \in \mathbb{N}$.

Question: Does there exist a set $S \subseteq V(G)$ of at most k vertices such that $G[V - S]$ satisfies property Π and for each vertex $v \in V(G)$, $|N(v) \cap S| \leq d$?

Using Π -FAIR VERTEX DELETION, we can define FAIR VERTEX COVER and FAIR FEEDBACK VERTEX SET where Π is the class of edgeless graphs and acyclic graphs respectively.

9.1.1 Previous Work and Deconstructing Hardness

Fair deletion problems were introduced by Lin and Sahni [112] where edges are deleted. Here, we require that the number of edges incident for every vertex in the graph to be bounded. This could also be viewed as minimizing the maximum degree of the graph restricted to the deleted edges. The authors showed that such problems are NP-complete. Later, Kolman et al. gave an XP algorithm for a generalization of fair edge deletion problems parameterized by treewidth when the property Π is expressible by a Monadic Second Order (MSO) formula.

The Π -FAIR VERTEX DELETION problem was introduced by Masařík and Toufar [130] where they studied the problem in the lens of logic. They first looked at Π -FAIR VERTEX DELETION where Π is expressible by a First Order formula also given as input. They showed that the problem is $W[1]$ -hard when parameterized by the sum of treedepth and the size of the minimum feedback vertex set of the graph. They also presented an FPT algorithm parameterized by the neighborhood diversity of the graph when Π can be expressed by an MSO formula given as input. Later, Knop, Masařík and Toufar [105] showed that Π -FAIR VERTEX DELETION where Π is expressible by an MSO formula with one free variable is FPT when parameterized by the twin cover number of the graph.

They also showed that FAIR VERTEX COVER is $W[1]$ -hard when parameterized by both treedepth and feedback vertex set of the input graph.

Since Π -FAIR VERTEX DELETION problems like FAIR VERTEX COVER are $W[1]$ -hard parameterized by the minimum feedback vertex set of the graph, they are $W[1]$ -hard parameterized by the treewidth t of the graph for $t \geq 1$ as the latter is a smaller parameter. But we note that in the reduction used in the proofs, the fairness factor d is in $\Omega(n)$. Inspired by the parameter ecology program of Fellows et al. [64], we believe that it is more natural to consider the fairness factor d also as a parameter in these problems.

9.1.2 Our Results

We first show that Π -FAIR VERTEX DELETION is FPT parameterized by $t + d$ if Π -VERTEX DELETION can be expressed by an MSO logic formula of constant length. This includes the fair version of a huge class of well-studied vertex deletion problems such as FEEDBACK VERTEX SET, ODD CYCLE TRANSVERSAL, CLUSTER VERTEX DELETION and CHORDAL VERTEX DELETION. Unfortunately, the FPT algorithms obtained above have a large exponential dependence on the parameter in the running time. Hence we study two classic problems FAIR VERTEX COVER and FAIR FEEDBACK VERTEX SET and give dynamic programming algorithms with much better running time parameterized by $t + d$. As a corollary, we get an FPT algorithm for FAIR FEEDBACK VERTEX SET parameterized by solution size as well. We remark that the standard reduction rules of FAIR FEEDBACK VERTEX SET do not seem to have an easy implementation to maintain the fairness constraint, and hence we do not know of other FPT algorithms for FAIR FEEDBACK VERTEX SET.

We then take a closer look at FAIR VERTEX COVER and more generally Π -FAIR VERTEX DELETION where the graph class Π has a finite forbidden family \mathcal{F} . We call the forbidden family \mathcal{F} a q -forbidden family if the vertex set of each graph in \mathcal{F} is of size at most q . In a

graph G , we say a subset $S \subseteq V(G)$ *hits* all graphs in \mathcal{F} when for all induced subgraphs H of G such that H is isomorphic to some member in \mathcal{F} , $S \cap V(H) \neq \emptyset$.

We now define Π -FAIR VERTEX DELETION where Π is a graph class having a q -forbidden family.

FAIR q -FORBIDDEN FAMILY VERTEX DELETION

Input: Given a graph $G = (V, E)$, a q -forbidden set \mathcal{F} where $q, k, d \in \mathbb{N}$.

Question: Does there exist a subset $S \subseteq V(G)$ of at most k vertices such that S hits all the occurrences of graphs in family \mathcal{F} in G and for each vertex $v \in V(G)$, $|N(v) \cap S| \leq d$?

We give a simple $\mathcal{O}^*(q^k)$ FPT algorithm for FAIR q -FORBIDDEN FAMILY VERTEX DELETION parameterized by solution size k . The more challenging task is to design a polynomial kernel as here again the standard reduction rules for q -FORBIDDEN FAMILY VERTEX DELETION do not keep track of fairness constraints. We reduce the problem instance to a MIN-ONES-SAT instance formula where all the clauses are constant-sized and each clause is either monotone or anti-monotone. We then show that this variant of MIN-ONES-SAT has a polynomial kernel. The latter result is of independent interest as in contrast, the general MIN-ONES-SAT problem even when all the clauses are of size 3 does not have a polynomial kernel [107], unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.

The FAIR VERTEX COVER problem can also be thought of as a special instance of FAIR q -FORBIDDEN FAMILY VERTEX DELETION with the q -Forbidden Set being the single edge K_2 . For FAIR VERTEX COVER we obtain a much better kernel through a different technique.

Finally, we complement our FPT results with some hardness results. We show that FAIR SET (defined below) and FAIR INDEPENDENT SET are $W[1]$ -hard even in 3-degenerate graphs, with fairness factor 1 when parameterized by solution size.

FAIR SET

Input: A graph $G = (V, E)$ and $k, d \in \mathbb{N}$.

Question: Does there exist a set $S \subseteq V(G)$ of at least k vertices such that for each vertex $v \in V(G)$, $|N(v) \cap S| \leq d$?

We note that the FAIR SET can be seen as a special case of (σ, ρ) DOMINATING SET [149] where the sets σ and $\rho = \{0, 1, \dots, d\}$. The (σ, ρ) DOMINATING SET is shown to be FPT when parameterized by treewidth when the sets σ and ρ are finite or cofinite [151] with running time $\mathcal{O}^*((st)^{s-2}s^t)$ where t is the treewidth of the graph and s is the number of states in the sets. As the number of states in the FAIR SET can be shown to be $d + 1$, the problem is FPT parameterized by sum of treewidth t and fairness factor d with running time $\mathcal{O}^*((d+1)t^{d-1}(d+1)^t)$.

A FAIR DOMINATING SET is a dominating set which is also a fair set. A problem very closely related to FAIR DOMINATING SET named $[1, j]$ DOMINATING SET has been studied recently [5] where it is shown to be $W[1]$ -hard in graphs of degeneracy $j + 1$ and FPT in nowhere dense graphs parameterized by the solution size. We note that in the $W[1]$ -hardness reduction, the $[1, j]$ dominating set is also a fair dominating set with fairness j and the FPT algorithm for nowhere dense graphs can be easily extended to FAIR DOMINATING SET.

Concerning the solvability of FAIR VERTEX COVER and FAIR FEEDBACK VERTEX SET for specific values of d , we note that FAIR VERTEX COVER and FAIR FEEDBACK VERTEX SET are NP-hard when $d = 3$ and $d = 4$ respectively as VERTEX COVER is NP-hard in cubic graphs [78] and FEEDBACK VERTEX SET is NP-hard on graphs with degree at most 4 [146]. For FAIR VERTEX COVER, we show that the problem is polynomial-time solvable when $d = 1$ or 2. For FAIR FEEDBACK VERTEX SET, we complete the picture by showing that the problem is NP-hard even when $d \in \{1, 2, 3\}$.

We end this section with the following observation, which essentially says that when the

parameter is the solution size, Π -FAIR VERTEX DELETION problems are interesting only when $d \leq k$.

Observation 9.1.1. *When $d \geq k$, Π -FAIR VERTEX DELETION is FPT when parameterized by solution size k whenever the corresponding Π -VERTEX DELETION problem (without the fairness constraint) is FPT when parameterized by solution size.*

This is because when $d \geq k$, the Π -FAIR VERTEX DELETION problem turns into the standard Π -VERTEX DELETION problem as every vertex has at most $k \leq d$ neighbors in the solution.

9.2 Preliminaries

We state the sunflower lemma which is widely applied in kernelization algorithms.

Definition 9.2.1 (Sunflower). *A sunflower with k petals and a core Y is a collection of sets S_1, \dots, S_k such that $S_i \cap S_j = Y$ for all $i \neq j$ and the sets $S_i \setminus Y$ are non-empty.*

Lemma 9.2.1 (Sunflower Lemma). *[45] Let \mathcal{A} be a family of sets (without duplicates) over a universe U such that each set in \mathcal{A} has cardinality exactly q . If $|\mathcal{A}| > q!(k-1)^q$, then \mathcal{A} contains a sunflower with k petals and can be computed in time polynomial in $|\mathcal{A}|$, $|U|$ and k .*

9.3 Π -FAIR VERTEX DELETION parameterized by treewidth + fairness factor

Here, we use the famous Courcelle's theorem which we state below.

Theorem 52 (Courcelle's theorem [44]). *Given an MSO formula ϕ , an n -vertex graph G and a tree decomposition of G of width t , there exists an algorithm that verifies whether ϕ*

is satisfied in G in time $f(|\phi|, t) \cdot n$ for some computable function f and $|\phi|$ denoting the length of encoding of ϕ as a string.

We show the following theorem.

Theorem 53. *The Π -FAIR VERTEX DELETION problem is FPT parameterized by the sum of treewidth and fairness factor if the corresponding Π -VERTEX DELETION problem can be expressed by an MSO formula of constant length.*

Proof. Let ϕ_1 be the constant length formula expressing the Π -VERTEX DELETION problem sentence. We can express the fairness property for a set of vertices with fairness factor d by the following MSO logic formula ϕ_2 .

$$\phi_2 := \exists S \forall u \in V \nexists v_1, \dots, v_{d+1} \in S \text{ such that } \{(u, v_1), \dots, (u, v_{d+1})\} \in E$$

The length of the formula ϕ_2 is linear in d . The Π -FAIR VERTEX DELETION problem can be expressed by MSO formula $\phi = \phi_1 \wedge \phi_2$ which is of length linear in d . Hence by Courcelle's theorem, the result follows. \square

Sentences of most of the Π -VERTEX DELETION problems like VERTEX COVER and FEEDBACK VERTEX SET can be expressed by a constant length MSO formula. Hence the above theorem gives FPT algorithms for these problems. But the running time of these FPT algorithms have huge exponents. So we focus on FAIR VERTEX COVER and FAIR FEEDBACK VERTEX SET and give better FPT algorithms using dynamic programming on tree decompositions.

Theorem 54. *FAIR VERTEX COVER can be solved in running time $\mathcal{O}^*(2^{\text{tw}}(d+1)^{3\text{tw}})$ on graphs of treewidth tw if a nice tree decomposition of width ω is given as input.*

Proof. Let $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ be the nice tree decomposition of the input graph G with treewidth tw . We use V_t to denote the set of vertices of G contained in the bags of the subtree rooted at t .

For a bag X_t of node t , we fix a set $Y \subseteq X_t$ and a partition $\mathcal{P} = (W_0, W_2, \dots, W_d)$ of X_t . We define $DP[t, Y, \mathcal{P}]$ as the size of the minimum fair vertex cover S of the graph $G[V_t]$ satisfies the following.

- $S \cap X_t = Y$.
- The set V_t is partitioned into sets C_0, \dots, C_d . For $i \in \{0, 1, \dots, d\}$, the set C_i contains vertices $v \in V_t$ such that $|N(v) \cap S| = i$. The sets C_i intersect in X_t in sets W_i .

We now provide recursive formulas to compute the values of the entries in DP .

- Leaf Node: Since $X_t = \emptyset$, the entry $DP[t, Y = \phi, \mathcal{P} = \emptyset] = 0$.
- Introduce Node: The node t has a single child node t' such that $X_t = X_{t'} \cup \{v\}$.
 - Case 1: $v \in Y$ and $v \in W_i$ where $i \neq d$.

$$DP[t, Y, \mathcal{P}] = 1 + DP[t', Y \setminus v, \mathcal{P}']$$

where \mathcal{P}' is obtained by removing v from W_i .

- Case 2: $v \notin Y$ and $v \in W_i$.

$$DP[t, Y, \mathcal{P}] = DP[t', Y, \mathcal{P}']$$

where \mathcal{P}' is obtained by removing v from W_i .

- Forget Node: The node t has a single child node t' such that $X_t = X_{t'} \setminus \{v\}$.

$$DP[t, Y, \mathcal{P}] = \min \left\{ \min_{\mathcal{P}'} DP[t', Y \cup \{v\}, \mathcal{P}'], \min_{\mathcal{P}'} DP[t', Y, \mathcal{P}'] \right\}$$

where \mathcal{P}' is obtained by adding v to one of the parts W_i .

- **Join Node:** The node t is a parent node of two children nodes t_1 and t_2 such that $X_t = X_{t_1} = X_{t_2}$.

$$DP[t, Y, \mathcal{P}] = \min_{\mathcal{P}_1, \mathcal{P}_2} \{DP[t_1, Y, \mathcal{P}_1] + DP[t_2, Y, \mathcal{P}_2] - |Y|\}$$

where \mathcal{P} is formed from compatible partitions \mathcal{P}_1 and \mathcal{P}_2 defined as follows. Look at a vertex $v \in X_t$. If $v \in W_i^1 \in \mathcal{P}_1$ and $v \in W_{i'}^2 \in \mathcal{P}_2$, then $v \in W_{i+i'} \in \mathcal{P}$ for $i+i' \leq d$. If $i+i' > d$ for some vertex v , then \mathcal{P}_1 and \mathcal{P}_2 are incompatible.

Running time: For each node t , there are at most 2^{tw} sets Y of X_t . There are $(d+1)^{\text{tw}}$ partitions \mathcal{P} of size $d+1$. Hence, the number of states per node is at most $2^{\text{tw}}((d+1))^{\text{tw}} \cdot n$. For a join node, for a state, we go over pairs of partitions \mathcal{P}_1 and \mathcal{P}_2 which is bounded by $(d+1)^{2\text{tw}}$. Hence, the overall running time is $\mathcal{O}^*(2^{\text{tw}}((d+1))^{3\text{tw}})$. \square

As a corollary of the above theorem, we show that the FAIR VERTEX COVER is FPT parameterized by the sum of chordal vertex deletion size and fairness factor of the graph. We make the following crucial observation.

Observation 9.3.1. *There does not exist a fair vertex cover with fairness factor d for a graph G with maximum clique size more than $d+1$.*

Proof. Suppose not. Let S be a fair vertex cover with fairness factor d for a graph G . Let us look at a clique C of size more than $d+1$ in G . Since S is a vertex cover, $|S \cap C| \geq |C| - 1 > d$. Hence, there exist a vertex $v \in C$ with more than d neighbors which are the vertices of $S \cap C$. This contradicts that S is a fair vertex cover with fairness factor d . \square

We have the following corollary.

Corollary 9.3.1. *FAIR VERTEX COVER is FPT parameterized by the sum of chordal vertex deletion size and fairness factor.*

Proof. We know that chordal graphs have a tree decomposition where each bag is a clique. By adding the chordal vertex deletion set of size k to each of the bags, we have a tree decomposition where each bag is a clique plus k vertices. If the maximum clique size of the graph is more than $d + 1$, using Observation 9.3.1, we can conclude that the given instance is a NO-instance. Else, we know that the maximum clique size is bounded by $d + 1$. Hence the treewidth of the graph is bounded by $k + d$. We now use the algorithm in Theorem 54 with running time $\mathcal{O}^*(2^{k+d}((d+1))^{3(k+d)})$. \square

Theorem 55. FAIR FEEDBACK VERTEX SET can be solved in running time $\mathcal{O}^*((\text{tw}(d+1))^{2\text{tw}})$ on graphs of treewidth tw if a nice tree decomposition of width tw is given as input.

Proof. Let $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ be the nice tree decomposition of the input graph G with treewidth tw .

For a bag X_t of node t , we set a partition $\mathcal{P} = (W_1, W_2, \dots, W_p, Y)$ of X_t . Also for $i \in [p]$, we define $(d+1)$ -sized partitions $\mathcal{Z}^i = (Z_0^i, Z_1^i, \dots, Z_d^i)$ of each sets W_i in the partition \mathcal{P} and partition $\mathcal{Z}^{p+1} = (Z_0^{p+1}, Z_1^{p+1}, \dots, Z_d^{p+1})$ of Y . Now we define $DP[t, \mathcal{P}, \mathcal{Z}]$ where $\mathcal{Z} = (\mathcal{Z}^1, \dots, \mathcal{Z}^{p+1})$ as the size of the minimum fair feedback vertex set S of the graph $G[V_t]$ satisfying the following.

- $S \cap X_t = Y$.
- $G[V_t] \setminus S$ is a forest containing p connected components C_1, \dots, C_p which intersect the bag X_t in non empty sets W_1, \dots, W_p respectively.
- For each set W_i , the subsets Z_j^i contain vertices that has exactly j neighbors to the solution S . Note that some of the sets Z_j^i could be empty as well.
- For set Y , the sets Z_j^{p+1} has vertices that has exactly j neighbors to the solution S . The set Z_j^{p+1} could be empty as well.

We now provide recursive formulas to compute the values of the entries in DP .

- Leaf Node: Since $X_t = \emptyset$, the entry $DP[t, \mathcal{P} = \phi, \mathcal{Z} = \emptyset] = 0$.
- Introduce Node: The node t has a single child node t' such that $X_t = X_{t'} \cup \{v\}$.

- Case 1: If $v \in Y$ and there are no vertex $u \in Z_d^i \cap N(v)$ for $i \in [p]$.

$$DP[t, \mathcal{P} = (W_1, \dots, W_p, Y), \mathcal{Z}] = 1 + DP[t', \mathcal{P}' = (W_1, \dots, W_p, Y \setminus v), \mathcal{Z}']$$

where \mathcal{Z} is obtained from \mathcal{Z}' by moving vertices $u \in Z_j^i \cap N(v)$ to set Z_{j+1}^i for $0 \leq j \leq d-1$ and keeping in Z_{j+1}^i otherwise.

- Case 2: If $v \in W_i$ and there are no vertices $u, w \in W_j \cap N(v)$.

$$DP[t, \mathcal{P} = (W_1, \dots, W_p, Y), \mathcal{Z}] = DP[t', \mathcal{P}' = (W_1, \dots, W_i \setminus v, \dots, W_p, Y), \mathcal{Z}']$$

where \mathcal{Z}' is obtained by removing the corresponding entry for v from \mathcal{Z} . Note that it is possible that $W_i = \{v\}$ as well in which case you remove the i^{th} part in \mathcal{P}' .

- Case 3: Otherwise $DP[t, \mathcal{P}, \mathcal{Z}] = \infty$.

- Forget Node: The node t has a single child node t' such that $X_t = X_{t'} \setminus \{v\}$.

$$DP[t, \mathcal{P}, \mathcal{Z}] = \min_{\mathcal{P}', \mathcal{Z}'} DP[t', \mathcal{P}', \mathcal{Z}']$$

where $\mathcal{P}' = (W_1, \dots, W_i \cup \{v\}, \dots, W_p, Y)$ for some $i \in [p]$ or

$\mathcal{P}' = (W_1, \dots, W_i, \dots, W_p, Y \cup \{v\})$. The minimum is taken over all \mathcal{Z}' which is obtained from \mathcal{Z} by updating $Z_j^i = Z_j^i \cup \{v\}$ for some $0 \leq j \leq d$.

- Join Node: The node t is a parent node of two children nodes t_1 and t_2 such that $X_t = X_{t_1} = X_{t_2}$.

$$DP[t, \mathcal{P}, \mathcal{Z}] = \min_{\mathcal{P}_1, \mathcal{P}_2, \mathcal{Z}_1, \mathcal{Z}_2} \{DP[t_1, \mathcal{P}_1, \mathcal{Z}_1] + DP[t_2, \mathcal{P}_2, \mathcal{Z}_2] - |Y|\}$$

where \mathcal{P}_1 and \mathcal{P}_2 are partitions such that \mathcal{P} can be obtained as the *acyclic merging* of \mathcal{P}_1 and \mathcal{P}_2 defined below. Let $\mathcal{P}_1 = (W_1^1, W_2^1, \dots, W_{p_1}^1, Y)$ and $\mathcal{P}_2 = (W_1^2, W_2^2, \dots, W_{p_2}^2, Y)$ be the partitions with Y , the part of the solution intersecting the bag X_t being common. If merging the two forests corresponding to \mathcal{P}_1 and \mathcal{P}_2 do not form any cycle and create a forest whose corresponding partition in X_t is \mathcal{P} , then \mathcal{P} is the *acyclic merging* of \mathcal{P}_1 and \mathcal{P}_2 .

The tuple \mathcal{Z} is formed as follows: Look at a vertex $v \in X_t \setminus Y$. It is present in sets Z_j^i and $Z_{j'}^{i'}$ corresponding to parts in $W_i^1 \in \mathcal{P}_1$ and $W_{i'}^2 \in \mathcal{P}_2$. If $j + j' > d$, the merge is invalid. Else the merge is valid and v is put in set $Z_{j+j'}^{i''}$ corresponding to part in $W_{i''} \in \mathcal{P}$. For vertex $v \in Y$ present in Z_j^{p+1} and $Z_{j'}^{p+1}$ corresponding to Y in \mathcal{P}_1 and \mathcal{P}_2 , we move it to $Z_{j+j'}^{p+1}$ if $j + j' \leq d$. Else the merge is invalid again.

Running time: For each node t , there are at most tw^{tw} partitions of \mathcal{P} of X_t . For each W_i , there are $(d+1)^{\text{tw}}$ partitions \mathcal{Z}_i . Hence, the number of states per node is $\text{tw} \text{tw}^{\text{tw}} \text{tw} (d+1)^{\text{tw}} = \mathcal{O}((\text{tw}(d+1))^{\text{tw}})$. Since at join node we go over pairs of states, overall running time is $\mathcal{O}^*((\text{tw}(d+1))^{3\text{tw}})$. \square

Corollary 9.3.2. FAIR FEEDBACK VERTEX SET is FPT parameterized by solution size k .

Proof. If $d > k$, then from Observation 9.1.1, the problem is FPT from the FPT algorithm for Feedback Vertex Set [106]. Else if the treewidth of the graph is at most $k+1$, then we first construct a tree decomposition of width $k' = 4(k+1) + 4$ in $\mathcal{O}(8^k \cdot n^2)$ time [45]. Then we apply Theorem 55 with this tree decomposition to get running time $\mathcal{O}^*((k'(d+1))^{3k'}) = k^{\mathcal{O}(k)}$. Else the treewidth of the graph is greater than $k+1$. But if there exists a feedback vertex set S of size k , then we can always get a tree decomposition with treewidth $k+1$ by adding S to each of the bags in the tree decomposition obtained for the forest $G \setminus S$. Hence if the treewidth of graph is greater than $k+1$, then we can conclude that there is no feedback vertex set of size k and return NO. \square

Corollary 9.3.3. FAIR FEEDBACK VERTEX SET is FPT parameterized by chordal vertex deletion size k .

Proof. We use similar arguments for FAIR VERTEX COVER. In a clique C , at least $|C| - 2$ vertices have to be in any feedback vertex set. Hence if the maximum clique of the graph is more than $d + 2$, we can find a vertex in the clique with $d + 1$ neighbors in the clique violating the fairness constraint. Hence, we can conclude that the input is a NO-instance.

The treewidth of the graph is hence bounded by $k + d + 1$. We apply Theorem 55 to get an algorithm with running time $((k + d + 1)d)^{\mathcal{O}(k+d+1)} \cdot n$. for FAIR FEEDBACK VERTEX SET. □

9.4 FAIR q -FORBIDDEN FAMILY VERTEX DELETION parameterized by solution size

9.4.1 FPT Algorithm

We first give a simple FPT algorithm for FAIR q -FORBIDDEN FAMILY VERTEX DELETION.

Theorem 56. FAIR q -FORBIDDEN FAMILY VERTEX DELETION can be solved in $\mathcal{O}^*(q^k)$ time.

Proof. We use the folklore result to enumerate all minimal q -forbidden family vertex deletion sets of size at most k . The algorithm uses the standard branching technique on vertex subsets of size at most q whose corresponding induced graph is a member of the forbidden family \mathcal{F} . This generates a search tree with at most q^k leaves. Every minimal q -forbidden family vertex deletion set of size at most k appears in one of the leaves of this tree.

Now for each such minimal q -forbidden family vertex deletion set X , we check if X is a d -fair set in polynomial time. If such a set X exists, we return YES-instance. Else we return NO-instance.

The running time of the algorithm is $\mathcal{O}^*(q^k)$ as the number of leaves of the search tree is at most q^k . □

Since FAIR VERTEX COVER is a special case of FAIR q -FORBIDDEN FAMILY VERTEX DELETION with the forbidden set being an edge, we have the following corollary.

Corollary 9.4.1. FAIR VERTEX COVER *can be solved in $\mathcal{O}^*(2^k)$ running time.*

9.4.2 Polynomial Kernel

The q -FORBIDDEN FAMILY VERTEX DELETION is known to have a kernel of size $\mathcal{O}(k^q)$. The problem can be easily reduced to the q -HITTING SET problem with the universe being $V(G)$ and the family containing all subsets $Q \subseteq V(G)$ such that $G[Q]$ is a member of \mathcal{F} . Using reductions rules based on the popular Sunflower Lemma, we may obtain a $\mathcal{O}(k^q)$ kernel q -HITTING SET. Similar reduction rules could be devised to obtain smaller instances of q -FORBIDDEN FAMILY VERTEX DELETION as well leading to a $\mathcal{O}(k^q)$ kernel.

We observed that the Sunflower Lemma based reduction rules used to obtain a kernel for q -FORBIDDEN FAMILY VERTEX DELETION do not work when we bring fairness constraints. This is because by deleting a vertex, we forget that only d neighbors of it are allowed to be in the solution. Hence we take a different approach by casting the problem as a special case of the well studied MIN ONES-SAT problem.

A clause of a formula in the conjunctive normal form(CNF) is *monotone* if all its literals are positive. If all its literals are negative, we call it *anti-monotone*. Let us define the following problem.

MIN-ONES-MONOTONE/ANTI-MONOTONE ℓ -SAT

Input: A CNF formula ϕ on n variables and m clauses such that all the clauses have at most ℓ variables and are either monotone or anti-monotone.

Question: Does there exist an assignment A with at most k variables set to true that satisfies ϕ ?

Theorem 57. MIN-ONES-MONOTONE/ANTI-MONOTONE ℓ -SAT parameterized by k has a polynomial kernel for constant ℓ .

Proof. Let $\phi = \phi_1 \wedge \phi_2$ where ϕ_1 is the conjunction of all the monotone clauses in ϕ and ϕ_2 is the conjunction of all the anti-monotone clauses in ϕ . Let a and b be the maximum size of the clauses in ϕ_1 and ϕ_2 , respectively.

Since ϕ_1 is a monotone formula, it can be treated as a set system with the universe being the variables and the family of sets being the monotone clauses. We have the following reduction rules.

Reduction Rule 10. Suppose that in the set system of the formula ϕ_1 , there exists a sunflower $\mathcal{S} = \{C_1, \dots, C_{k+1}\}$ with the core set of positive literals $Y = \{x_{y_1}, \dots, x_{y_p}\}$. If Y is empty, then we return NO. Else in the formula ϕ we remove all the clauses C_i with $i \in [k+1]$ and add a clause $C_Y = x_{y_1} \vee \dots \vee x_{y_p}$. If $|Y| = 1$, we set the corresponding variable to 1 in all the clauses in the formula.

Claim 9.4.1. Reduction Rule 10 is safe and can be performed in polynomial time.

Proof. We prove this by showing that the formula ϕ is satisfiable by setting at most k variables to 1 if and only if the formula ϕ' obtained after applying the reduction rule is satisfiable with setting at most k variables to 1.

Let A be the assignment of ϕ by setting at most k variables to 1. We know that in A , one of the variables in Y has to be set to 1 as otherwise we need to set $k+1$ distinct variables of

monotone clauses C_i not in Y to 1 violating the assumption that at most k variables are set to 1. This also means that Y is non-empty as if it is empty we get a NO-instance of the problem. Hence in formula ϕ' , the clause C_Y is set to true. Since ϕ' is formed by removing the clauses C_i and adding C_Y , ϕ' is also satisfied by assignment A with at most k variables set to 1.

For the converse, let A' be the assignment that sets ϕ' satisfiable by setting at most k variables to 1. Since one of the variables in C_Y is set to 1 and all clauses C_i contains all the variables in Y , all the clauses C_i in formula ϕ are set to true by A' . Hence ϕ is also satisfied with at most k variables set to 1.

Note that when the core C_Y is a singleton set, we know the corresponding variable is set to 1. □

Reduction Rule 11. *Let x_v be a variable that is present only in negative form in ϕ , then delete all the clauses containing x_v .*

Claim 9.4.2. *Reduction Rule 11 is safe.*

Proof. Since the variable appears only in the negative form, it appears in ϕ_2 consisting of anti-monotone clauses. Now since all the clauses containing x_v can be set to true by setting x_v to false, we can delete all these clauses and get a formula ϕ' which is also satisfiable with the same number of variables set to true in ϕ . □

Claim 9.4.3. *Let integers a and b be the number of variables in the monotone and antimotone clauses of ϕ . If Reduction Rules 10 and 11 are not applicable, then ϕ has $\mathcal{O}((a!k^a \cdot a^2)^b)$ clauses.*

Proof. Look at the subfamily of monotone clauses of size $a' \in [a]$ in ϕ . If the number of monotone clauses $N > a'!k^{a'}$, then by Sunflower Lemma 9.2.1, there exists a sunflower in the family. Hence we can apply Reduction Rule 10 which is not the case. Hence the

number of monotone clauses is at most $\sum_{a'=1}^a a'!k^{a'} \leq a!k^a \cdot a$. Let M be the set of variables appearing in these clauses. We have $|M| \leq a!k^a \cdot a^2$.

Now we look at the anti-monotone clauses of ϕ . There are at most $\binom{M}{b}$ clauses here containing only variables in M . In the rest of the clauses there exists a variable that is occurring only in the anti-monotone clauses of ϕ and hence only in negative form in ϕ . Reduction rule 11 would have removed all such clauses.

Hence, the number of clauses in ϕ is bounded by $a!k^a \cdot a + \binom{a!k^a \cdot a^2}{b} = \mathcal{O}((a!k^a \cdot a^2)^b)$. \square

When $a, b \leq \ell$ are constants, the input formula size is polynomial in k . Thus we have a kernel of size $\mathcal{O}((a!k^a \cdot a^2)^b)$ \square

We now claim that FAIR q -FORBIDDEN FAMILY VERTEX DELETION has a polynomial kernel by giving a parameterized reduction to MIN-ONES-MONOTONE/ANTI-MONOTONE ℓ -SAT problem.

Theorem 58. *There is a polynomial kernel for FAIR q -FORBIDDEN FAMILY VERTEX DELETION parameterized by the solution size k when both q and the fairness factor d are constants.*

Proof. Let $l = \max\{q, d + 1\}$. Let (G, \mathcal{F}, k, d) be an instance FAIR q -FORBIDDEN FAMILY VERTEX DELETION. Let \mathcal{F}' be the family of subsets $Q \subseteq V(G)$ such that $G[Q] \in \mathcal{F}$. We construct an instance of MIN-ONES-MONOTONE/ANTI-MONOTONE l -SAT which is the formula ϕ as follows:

For each vertex $u \in V$, we have a variable x_u . We define two types of clauses:

- Monotone clauses: For each set $S \in \mathcal{F}'$ with $S = \{u_1, \dots, u_q\}$, we define the clause $C_S = x_{u_1} \vee \dots \vee x_{u_q}$.
- Anti-monotone clauses: For each vertex $u \in V(G)$, we look at the open neighborhood

set $N(u)$. For all sets $D \subseteq N(u)$ of size $d + 1$, say $D = \{v_1, \dots, v_{d+1}\}$, we construct a clause $C_D = \bar{x}_{v_1} \vee \dots \vee \bar{x}_{v_{d+1}}$.

If we look at any $d + 1$ -sized set from the open neighborhood of any vertex $u \in V$, at least one of the vertices cannot be in the solution as otherwise it violates the fairness of vertex u . This is captured by the anti-monotone clauses C_D .

The formula ϕ is the conjunction of all the clauses C_S and C_D . Note that since $|N(u)| \leq n - 1$, the number of clauses is $\mathcal{O}(m + n^{d+1} \cdot n)$ which is polynomial in n and m for a constant d . Let the formula formed by the conjunction of all the monotone clauses be ϕ_1 and by all the anti-monotone clauses be ϕ_2 . Note that here $a = q$ and $b = d + 1$.

It can be easily shown that (G, \mathcal{F}, k, d) is a YES-instance of FAIR q -FORBIDDEN FAMILY VERTEX DELETION if and only if (ϕ, k) is a YES-instance of MIN-ONES-MONOTONE/ANTI-MONOTONE ℓ -SAT. We use Theorem 57 to obtain a kernel (ϕ, k') of MIN-ONES-MONOTONE/ANTI-MONOTONE SAT with the formula ϕ' size being $\mathcal{O}((q!k^q \cdot q^2)^{d+1})$. Since both the problems are NP-complete, there is a polynomial time reduction from MIN-ONES-MONOTONE/ANTI-MONOTONE SAT back to FAIR q -HITTING SET. We use this reduction on ϕ' to get a FAIR q -HITTING SET instance of size $|\phi'|^{\mathcal{O}(1)} = ((q!k^q \cdot q^2)^{d+1})^{\mathcal{O}(1)}$ which is polynomial in k when q and d are constants. □

As FAIR VERTEX COVER is a special case of FAIR q -HITTING SET with $q = 2$, we have the following corollary.

Corollary 9.4.2. FAIR VERTEX COVER parameterized by solution size k has a kernel of size $k^{\mathcal{O}(d)}$.

9.4.3 Improved Kernel for FAIR VERTEX COVER parameterized by solution size

For FAIR VERTEX COVER, we observe that we can get an improved kernel by modifying the classical Buss kernel [31] for VERTEX COVER.

We apply the following reduction rules in sequence, only once.

Reduction Rule 12. *Delete all isolated vertices in (G, k) .*

The above reduction rule is safe as isolated vertices do not cover any edge.

Reduction Rule 13. *Let H be the set of vertices in G having degree greater than d . If $|H| > k$ or H is not a fair set, then return NO-instance. Else delete all the isolated vertices in $G[V \setminus H]$. Add $d + 1$ many pendant vertices adjacent to each $v \in H$. Return the resulting instance (G', k) .*

Lemma 9.4.1. *Reduction rule 13 is safe.*

Proof. Consider an instance (G, k) of FAIR VERTEX COVER. We claim that (G, k) is an YES instance if and only if (G', k) is an YES instance.

Let Z be a minimal solution of (G, k) . First, we claim that every vertex in H should be in Z . For contradiction let $v \in H$ such that $v \notin Z$. Then we know that $N(v) \subseteq Z$ as otherwise the edges incident to v are not covered. But this violates the fact that Z is a fair set as v will have $d + 1$ neighbors in Z . Hence, if H is not a fair set in G , there is no solution of size at most k .

Now consider the graph $G[V - H]$ with an isolated vertex $v \in V \setminus H$. Suppose $v \in Z$. From Reduction Rule 12, we know that v is not an isolated vertex in G . Hence, v is a neighbor to some vertex in H . We proved that $H \subseteq Z$. Hence, the edges incident on v are covered by vertices in H . Also since the degree of v is at most d , any solution in G' will not violate the

fairness condition of v . Thus, $Z - v$ is also a solution to (G, k) contradicting the minimality of Z .

Hence we can assume that all the vertices of Z are present in G'' which is the graph formed from G by removing isolated vertices in $G[V - H]$. Since G'' is a subgraph of G and fairness is preserved in subgraphs, $Z \setminus H$ is also a solution in G'' .

The graph G' is formed by adding $d + 1$ pendant vertices to each vertex $v \in H$ of the graph G'' . Since $H \subseteq Z$, Z is a vertex cover in G' as well. The neighborhood of the graph changes only for vertices in H . Hence we only need to verify that fairness is preserved by Z for the vertices H plus the newly added vertices. Since all the newly added vertices have only one neighbor in Z , fairness is preserved for them as $d \geq 1$. Since none of the newly added vertices are in Z , fairness is preserved for vertices in H . Hence we can conclude that Z is also a solution for the instance (G', k) .

For the converse, let Y be a solution for (G', k) . We claim that Y is a solution for (G, k) as well. Since there are $d + 1$ many pendant vertices on each $v \in H$, $H \subseteq Y$ as otherwise fairness is violated for v in G' . The graph G can be obtained from G' by removing the $d + 1$ pendant vertices of each vertex in H and adding back the isolated vertices in $G[V \setminus H]$. Since these isolated vertices are adjacent to H alone and have degree at most d , we can conclude that Y is a fair vertex cover in G as well. \square

Reduction Rule 14. *Let (G, k) be an input instance on which Reduction Rules 12 and 13 are not applicable. Let H be the set of vertices in G having degree greater than d . If there are more than $k \cdot d$ edges in $G[V \setminus H]$, then return NO.*

Lemma 9.4.2. *Reduction rule 14 is safe.*

Proof. Since every vertex in $V \setminus H$ has degree at most d , it can cover only at most d edges in $G[V \setminus H]$. Hence any set of k vertices can cover at most $k \cdot d$ edges in $G[V \setminus H]$. If there are more than $k \cdot d$ edges in $G[V \setminus H]$, then we can conclude that there is no vertex cover of size k in $G[V \setminus H]$ and therefore in the supergraph G . \square

Reduction Rule 15. Let (G, k) be an input instance on which Reduction Rules 12, 13 and 14 are not applicable. If G has more than $3kd + 2k$ vertices or $2k^2d + k^2 + 2kd + k$ edges, then return NO.

Lemma 9.4.3. Reduction rule 15 is safe.

Proof. Let (G, k) be an instance of FAIR VERTEX COVER obtained after applying Reduction rules 12, 13 and 14. Let H be the set of vertices in G having degree greater than d . Let N be the set of pendant vertices adjacent to some $v \in H$. Since the vertices in H belong to any solution of G , $|H| \leq k$. Since we added at most $d + 1$ many pendant vertices adjacent to each $v \in H$, we have $|N| \leq k(d + 1)$.

We claim that if (G, k) is a YES instance, $G \setminus (H \cup N)$ has at most $2kd$ many vertices. Since Reduction Rule 14 is no longer applicable and N is the set of isolated vertices in $G \setminus H$, there are at most kd edges in $G \setminus (H \cup N)$. Also, after the application of reduction rule 13, there are no isolated vertices in $G' \setminus (H \cup N)$. Hence there are at most $2kd$ many vertices in $G \setminus (H \cup N)$.

Thus the vertex set size of G is bounded by $2kd + |N| + |H| \leq 2kd + k(d + 1) + k = 3kd + 2k$.

We now bound the number of edges in G . Since Reduction Rule 14 is no longer applicable, there are at most kd edges in $G \setminus H$. Every other edge in G has one endpoint in H . Each vertex in $v \in H$ can be adjacent to all vertices in $V \setminus N$ and $d + 1$ vertices in N . Hence the number of edges adjacent to each $v \in H$ is bounded $2kd + k + d + 1$. Since $|H| \leq k$, the number of edges incident to k is bounded by $(2kd + k + d + 1)k$. Therefore, the number of edges in G is bounded by $(2kd + k + d + 1)k + kd = 2k^2d + k^2 + 2kd + k$.

Thus if G has more than $3kd + 2k$ vertices or $2k^2d + k^2 + 2kd + k$ edges, we conclude that we are dealing with a NO instance. □

This leads to our kernel result for FAIR VERTEX COVER.

Theorem 59. There exists a kernel for FAIR VERTEX COVER with $\mathcal{O}(kd)$ vertices and

$\mathcal{O}(k^2d)$ edges.

9.5 Hardness Results

9.5.1 W[1]-Hardness for Fair Set

Theorem 60. FAIR SET with $d = 1$ is W[1]-hard when parameterized by solution size k for graphs with degeneracy three.

Proof. We give a reduction from the MULTICOLORED INDEPENDENT SET problem known to be W[1]-hard [45] defined as follows.

MULTICOLORED INDEPENDENT SET

Input: A graph $G = (V, E)$ and partition of (V_1, \dots, V_k) of V for $k \in \mathbb{N}$.

Question: Does there exist a set $S \subseteq V$ of k vertices such that S forms an independent set and for each vertex V_i , $|V_i \cap S| = 1$?

Let (G, V_1, \dots, V_k) be the MULTICOLORED INDEPENDENT SET instance. Without loss of generality, assume that $G[V_i]$ is an independent set for all $i \in [k]$. We construct an instance $(G', k+2, 1)$ of FAIR SET with $d = 1$ as follows:

We start constructing G' with the same vertex set of G . For each class V_i , we introduce a vertex v_i and make it adjacent to all the vertices in V_i . For each edge $e_j = (u, v) \in E(G)$, we add a vertex e_j in G' and add edges (u, e_j) and (e_j, v) . We also add a vertex s adjacent to all edge vertices e_j for $j \in [m]$ and the vertices v_1, v_2, \dots, v_k . Finally we add vertices t adjacent to s and t' adjacent to t . Refer to Figure 9.1.

Claim 9.5.1. (G, V_1, \dots, V_k) is a yes instance for MULTICOLORED INDEPENDENT SET if and only if $(G', k+2, 1)$ is a yes instance for FAIR SET.

Proof. In the forward direction, let $X = \{x_1, \dots, x_k\}$ be a multicolored independent set in G with $x_i \in V_i$. Then we claim that $X' = X \cup \{t, t'\}$ is a fair set in G' . For a vertex $v \in V_i$,

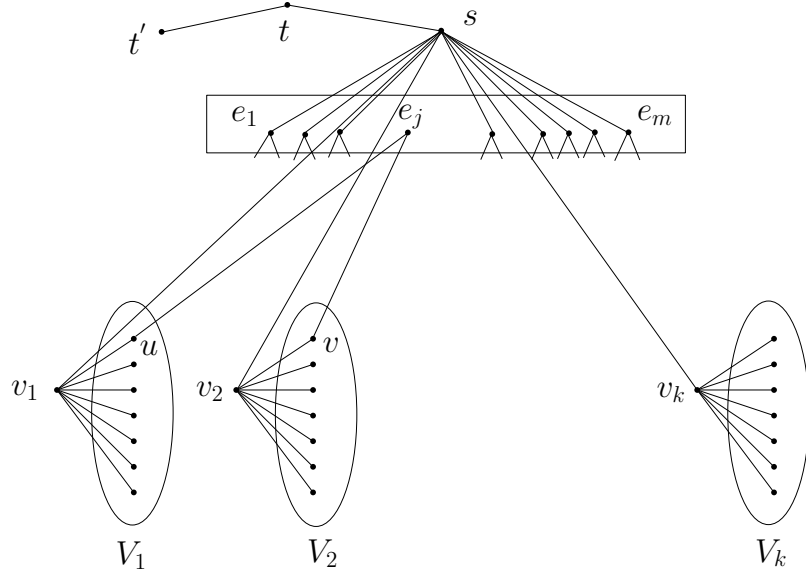


Figure 9.1: Construction of FAIR SET instance with $d = 1$ from MULTICOLORED INDEPENDENT SET instance in Theorem 60

$|N(v) \cap X'| \leq 1$. For vertex v_i , $N(v_i) \cap X' = \{x_i\}$. For an edge vertex e_j , $|N(e_j) \cap X'| \leq 1$ as X is an independent set in G . For vertex s , $N(s) \cap X' = \{t\}$, for vertex t , $N(t) \cap X' = \{t'\}$ and vertex t' , $N(t') \cap X' = \{t\}$. Hence X' is a fair set.

Conversely, let Y be a fair set in G' of size $k + 2$. Since the vertex v_i is adjacent to all the vertices in V_i , we have $|Y \cap V_i| \leq 1$ as otherwise fairness of v_i is violated. Since vertex s is adjacent to all the vertices e_j , v_i and vertex t , for $j \in [m]$, we have $|Y \cap \{e_1, \dots, e_m, v_1, \dots, v_k, t\}| \leq 1$. Also since t is adjacent to s and t' , we have $|Y \cap \{s, t'\}| \leq 1$. Hence Y can contain at most 2 vertices apart from the vertices from V_1, \dots, V_k , the original vertices of G . Since $|Y| = k + 2$, we can conclude that $|Y \cap V_i| = 1$ for $i \in [k]$ and $|Y \cap \{e_1, \dots, e_m, v_1, \dots, v_k, s, t, t'\}| = 2$. Let $Y' = \{y_1, \dots, y_k\}$ be the vertices in $Y \cap V_i$. We claim that Y' is an independent set in G . Suppose there is an edge between y_i and $y_{i'}$. Then fairness is violated in the corresponding vertex e_j in G' . Hence the claim follows. \square

Now we look at the degeneracy of the graph G' . We give the degeneracy order where we first put edge vertices e_1, \dots, e_m , then all the vertices in V_1, \dots, V_k and then vertices $v_1, \dots, v_k, s, t, t'$ in that order. It can be verified that the degeneracy of the graph is 3 from this order. \square

A fair independent set is an independent set which is also a fair set. The reduction in Theorem 60 can be slightly modified to give a $W[1]$ -hardness result for FAIR INDEPENDENT SET problem where we look for a fair independent set of size k .

Theorem 61. *The FAIR INDEPENDENT SET problem is $W[1]$ -hard parameterized by solution size k for graphs with degeneracy three.*

9.5.2 NP-Hardness Dichotomy of FAIR VERTEX COVER and FAIR FEEDBACK VERTEX SET

9.5.2.1 FAIR VERTEX COVER

Since VERTEX COVER is NP-hard on subcubic graphs [78], we know that FAIR VERTEX COVER is NP-hard when $d \geq 3$. We complete the picture by showing that the problem is polynomial-time solvable when $d = 1$ and $d = 2$.

Theorem 62. FAIR VERTEX COVER is polynomial-time solvable when $d = 1$.

Proof. Let S denote the set of all vertices of the graph G with degree more than 1. All the vertices $v \in S$ must be in any solution of FAIR VERTEX COVER as otherwise the fairness of v is violated. Hence if S is not a fair set, return NO . Look at the graph $G \setminus S$. Since the vertices in $G \setminus S$ have degree at most 1, it consists of isolated vertices and edges. Note that the endpoints of the isolated edges have no neighbors to S as well as they must have degree 1 in G . Let us create a set T by picking an arbitrary vertex from every isolated edge. We can easily see that $T \cup S$ is the minimum sized FAIR VERTEX COVER in G . \square

We now show that FAIR VERTEX COVER is polynomial-time solvable when $d = 2$ by observing that after all the vertices that are forced to go into the solution are picked, the problem boils down to a matching problem.

Theorem 63. FAIR VERTEX COVER is polynomial-time solvable when $d = 2$.

Proof. Let S denote the set of all vertices of the graph G with degree more than 2. All the vertices $v \in S$ have to go into the solution of FAIR VERTEX COVER as otherwise the fairness of v is violated. Hence if S is not a fair set, return NO . Look at the graph $G \setminus S$. Since the vertices in $G \setminus S$ has degree at most two, it has isolated vertices, paths and cycles. Note that since all the vertices of a cycle have degree two, there is no edge from any vertex in the cycle to S . Hence we can arbitrarily pick alternate vertices of the cycles in $G \setminus S$ into the solution.

Look at a path $P = (v_1, v_2, \dots, v_l)$ in $G \setminus S$ with $l > 2$. Since all the internal vertices of P have degree two, there are no edges from any such vertex to S . If l is odd, picking all vertices v_{2i} will cover all the edges of P . If l is even, picking all vertices v_{2i} other than v_l and the vertex v_{l-1} will cover all the edges of P . Note that in both cases, we pick only the internal vertices of P . Hence no fairness constraints are violated by picking these vertices.

Hence all the edges of G are covered except isolated edges in $G \setminus S$. Look at any isolated edge (u, v) . If u or v does not have any edges to S , then pick the corresponding vertex into the solution. Hence assume that both u and v have edges to S . Since u and v has degree at most two, they have a unique neighbor in S . Let us denote them as n_u and n_v .

Note that $G[S]$ has degree at most two as otherwise fairness is violated for some vertex. If n_u is a degree two vertex in $G[S]$, then the corresponding vertex u cannot go into the solution as otherwise the fairness of n_u is violated. We have the same conclusion for n_v . Thus, if exactly one of n_u or n_v , say n_u is of degree two, then we add v to the solution.

We can now conclude that either both n_u and n_v are isolated vertices in $G[S]$ or are endpoints of some path in $G[S]$.

Let A denote the isolated edges (u, v) in $G \setminus S$ remaining to be covered and B denote set of vertices n_u where u is an endpoint of these edges. The problem reduces to picking exactly one endpoint of each isolated edge in $G \setminus S$ such that for all vertices $v \in B$, the number of vertices of $N(v)$ picked is at most

- one when v is an endpoint of a path in $G[S]$ or
- two if v is an isolated vertex in $G[S]$ respectively.

To solve this problem, we construct a bipartite graph $H = (A, B')$ from G with $B' = B \cup I$ where I is a copy of all the isolated vertices in $G[S]$ which are present in B . We add edges (a, b) for $a \in A$ and $b \in B'$ when $(a_0, b_0) \in E$ where a_0 is one of the endpoints of the edge a and $b_0 \in B$ is the corresponding vertex in S .

Claim 9.5.2. *We claim that there is a matching saturating A in H if and only if there is a solution for FAIR VERTEX COVER in G .*

Proof. Let M be the matching saturating A in H if it exists. We construct a solution R that covers the isolated edges maintaining the fairness as follows: for each matched edge (e, w) where $e = (u, v)$, if (u, w) is an edge, we add u to R , else we add v to R . Note that fairness of no vertices in $V \setminus S$ is violated by adding R to the solution as they are from isolated edges. Also note that since all the vertices of isolated edges in A have degree exactly one into S , all vertices in $u \in R$ are neighbors only to its matched vertex w in the set S . Hence for all the vertices $w \in B$ which are endpoints of paths in $G[S]$, the fairness constraint is maintained as $|N(w) \cap R| \leq 1$. For the isolated vertices w in S , $|N(w) \cap R| \leq 2$ as there are two copies of w in B' . Since fairness constraint is satisfied for all the vertices in S as well, we have a FAIR VERTEX COVER in G .

To prove the converse, let us look at an optimal fair vertex cover R' in G . We know that $S \subseteq R'$. Each $a \in A$ correspond to some isolated edge (u, v) in $G - S$. To cover the edge, one of the endpoints is in R' . For each endpoint u in any isolated edge, there is a neighbor $n_u \in S$. Any vertex in R' that is not an isolated vertex in $G[S]$ is adjacent to at most one vertex in $(V \setminus S) \cap R'$. Any vertex in R' that is an isolated vertex in $G[S]$ is adjacent to at most two vertices in $(V \setminus S) \cap R'$.

We now try to construct a matching M saturating A as follows. For each $a \in A$ with the corresponding edge (u, v) , let $u \in R'$. We add the edge (a, n_u) to M . The set M is not a

matching only when there exist a vertex in $r \in R'$ that is part of two edges (a_1, r) and (a_2, r) for $a_1, a_2 \in A$. But this can happen only when r is an isolated vertex in $G[S]$. But in this case, there is a copy $r' \in I$ of r for the graph H . We remove the edge (a_2, r) and add the edge (a_2, r') to M . It is easy to see that M is a matching saturating A after we do the above process for all isolated vertices of $G[S]$ part of M . \square

Hence we have a polynomial-time algorithm for FAIR VERTEX COVER with $d = 2$ since we can find the above matching if it exists in polynomial time. \square

9.5.2.2 FAIR FEEDBACK VERTEX SET

Since FEEDBACK VERTEX SET is NP-hard on graphs with degree at most 4 [146], we know that FAIR FEEDBACK VERTEX SET is NP-hard when $d \geq 4$. We complete the picture by showing that the problem is NP-hard when $d \in \{1, 2, 3\}$.

Theorem 64. FAIR FEEDBACK VERTEX SET is NP-hard when $d \in \{1, 2, 3\}$.

Proof. We give the proof for $d = 1$. The proof for $d = 2$ and $d = 3$ are similar with slight modifications.

We give a reduction from 3-SAT. Given a 3-SAT instance formula ϕ with variables v_1, \dots, v_n and clauses C_1, C_2, \dots, C_m , we construct a FAIR FEEDBACK VERTEX SET instance graph G as follows:

For each clause $C_i = u \vee v \vee w$, we create a triangle of three vertices u_i, v_i, w_i corresponding to the literals in the clause. We then subdivide each edge in the triangle by adding three new vertices. That is, for the pair of literals (u_i, v_i) , we construct a path P of five vertices $u_i, e_i^{11}, e_i^{21}, e_i^{31}, v_i$. Similarly, we construct paths for pairs v_i, w_i using vertices e_i^{12}, e_i^{22} and e_i^{32} and w_i, u_i with vertices e_i^{13}, e_i^{23} and e_i^{33} . We create new vertices d_i , private to each clause and attach the middle vertices $e_i^{21}, e_i^{22}, e_i^{23}$ to d_i . Finally, we add a self loop to vertices d_i .

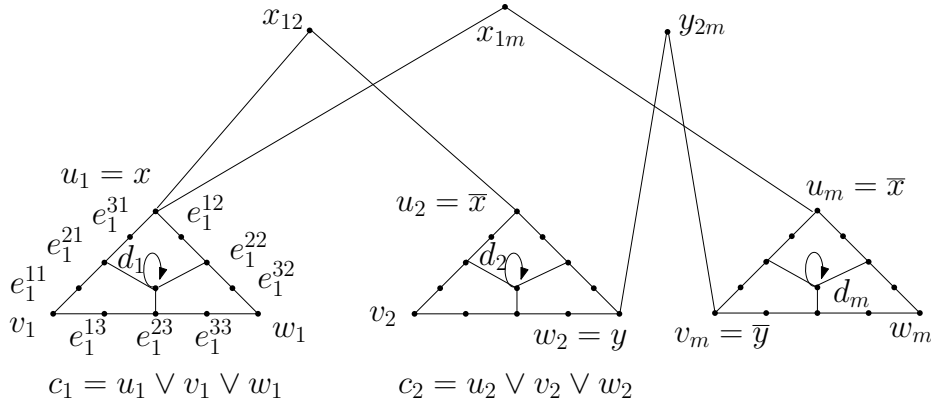


Figure 9.2: Construction of FAIR FEEDBACK VERTEX SET instance with $d = 1$ from 3-SAT instance in Theorem 64

Now for a variable x , if x occurs positively in clause C_i and negatively in clause C_j , we create a new vertex $x_{i,j}$ and add edges to the vertex corresponding to literals corresponding to x in C_i and C_j . We do so over all variables and all pairs of clauses.

Claim 9.5.3. *The formula ϕ is satisfiable if and only if there exist a fair feedback vertex set with $d = 1$ in G .*

Proof. Let A be a satisfying assignment to ϕ . We add all vertices in the triples corresponding to clauses where the corresponding literal is set to true by A into a set S . Also add all vertices d_i for all $1 \leq i \leq m$. We claim S is a fair feedback vertex set with $d = 1$.

First, we prove that S is a feedback vertex set. Suppose that there exists a cycle C in $G \setminus S$. Every cycle corresponding to clauses has a vertex corresponding to one of the literals in S by definition. Also all vertices d_i are in S . Hence it can be seen from the construction of G that the cycle C has to contain one of the vertices x_{ij} . Also, C has to contain both the two neighbors of x_{ij} that correspond to literal x and its complement. By definition of S , one of these two vertices must be in S as the corresponding literal is true. Hence no such cycle exists.

Now we prove that S is a fair set with fairness 1. Let us focus on the triple of vertices corresponding to a clause. Notice that none of its neighbors are in S . For the five length path u, e^1, e^2, e^3, v from two vertices u to v of a clause triplet, we see that fairness is preserved

for the vertices e^1 and e^3 as at most one of its neighbors (which is one of u or v) is in S . For vertices e^2 , again only one of its neighbors d_i is in S . The same goes for d_i with the neighbor being itself. Finally, we look at vertices x_{ij} . Since only one of the literal corresponding to variable x can be true, only one of its neighbors to clauses C_i and C_j is in S . Hence S is a fair feedback vertex set with fairness 1.

In the converse, let us look at a fair feedback vertex set X in G . All the vertices $d_i \in X$ as they have self-loops. Now for every five length path u, e^1, e^2, e^3, v from u to v , as vertices $d_i \in X$, the fairness of vertices e^2 is tight. Hence vertices e^1 and e^3 cannot be in X . Also since for vertices d_i the fairness is tight, the vertices $e^2 \notin X$. Thus to hit the cycle corresponding to the clauses, one of the literal vertices has to be in X . Now to preserve the fairness of vertices x_{ij} , for a variable x , it cannot be that both the vertex corresponding to the positive literal x and for the negative literal x be in the set X . Hence for all variables x , either all vertices in X are positive or all are negative. Corresponding to this, we create an assignment A . Since the cycle corresponding to every clause is hit by X , the assignment is a satisfying assignment to ϕ . □

This proves the theorem. □

9.6 Conclusion

We initiated a systematic study on various Π -FAIR VERTEX DELETION problems under various parameterizations. An open problem is to give a polynomial kernel for FAIR FEEDBACK VERTEX SET parameterized by solution size. Also finding FPT algorithms for other Π -FAIR VERTEX DELETION problems like FAIR ODD CYCLE TRANSVERSAL remains open.

Part IV

Conclusion

Chapter 10

Conclusion and Future Directions

In this thesis, we studied vertex deletion problems in three different directions. In the first direction (Chapters 3 and 4), we looked at vertex deletion to scattered graph classes and provided several FPT algorithms. The existence of a polynomial kernel for most cases of scattered version of problems is open. Exploring fixed-parameter tractability of scattered versions of edge deletion and edge contraction problems is also worth looking at. Giving faster FPT algorithms for pairs (Π_1, Π_2) of scattered classes that do not have finite forbidden pairs (such as (Chordal, Bipartite)) is another future direction.

In the second direction, we look at several deletion distance parameterizations. In Chapter 5, we observed the issues that come up from assuming that a modulator is given as input for deletion distance parameterizations. We provided FPT algorithms for several problems parameterized by deletion distance to chordal graphs without assuming modulator is given as input. In the future, we hope to see similar deletion distance parameterization results in the case when finding the modulator is ‘harder’ than solving problems using the modulator. Some such parameter examples are deletion distance to planar graphs and perfect graphs. The problem of PLANAR VERTEX DELETION can be solved in $k^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ time [96] while PERFECT VERTEX DELETION is $W[2]$ -hard [84]. The VERTEX COVER problems has a PTAS in planar graphs [12] and is polynomial-time solvable in perfect graphs [80]. The

first open problem is to give an approximation scheme in $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ time for VERTEX COVER with the parameter being deletion distance to planar graphs without the modulator assumption. The second open problem is to give a $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ time algorithm for VERTEX COVER with the parameter being deletion distance to perfect graphs without the modulator assumption.

In Chapter 6, we provided an FPT algorithm for $(n - k)$ LIST COLORING using a deletion distance parameterization for the same as a subroutine. In a later work, Gutin et al. [81] provided a kernel for the problem with $\mathcal{O}(k^2)$ vertices and colors. In Chapter 7, we looked at several dominating set problems parameterized by deletion distance to cluster and split graphs and provided FPT and kernel bounds for the same. Designing FPT algorithms for variants of DOMINATING SET where the parameter is deletion distance to a graph class (other than cluster or split graphs) where the variant is solvable in polynomial time is open.

In the third direction, we looked at vertex deletion problems where we want the solution set to satisfy additional constraints. In Chapter 8, we looked at CONFLICT FREE SET COVER which generalizes conflict-free version of several vertex deletion problems and provides FPT and kernel bounds under different parameterizations. An open problem is to identify a general characterization for the graph classes of $G_{\mathcal{F}}$ when a CONFLICT-FREE SET COVER variant studied becomes FPT. Finally, in Chapter 9, we looked at fair vertex deletion problems where we want our solution to also form a fair set and provide FPT and kernel results under different parameterizations. Open problems include giving a polynomial kernel for FAIR FEEDBACK VERTEX SET parameterized by solution size and finding FPT algorithms for other Π -FAIR VERTEX DELETION problems like FAIR ODD CYCLE TRANSVERSAL.

Bibliography

- [1] Akanksha Agrawal, Pallavi Jain, Lawqueen Kanesh, Daniel Lokshantov, and Saket Saurabh. Conflict free feedback vertex set: A parameterized dichotomy. In *LIPICs-Leibniz International Proceedings in Informatics*, volume 117. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
- [2] Akanksha Agrawal, Lawqueen Kanesh, Daniel Lokshantov, Fahad Panolan, MS Ramanujan, Saket Saurabh, and Meirav Zehavi. Deleting, eliminating and decomposing to hereditary classes are all fpt-equivalent. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1976–2004. SIAM, 2022.
- [3] Akanksha Agrawal, Daniel Lokshantov, Pranabendu Misra, Saket Saurabh, and Meirav Zehavi. Polylogarithmic approximation algorithms for weighted-f-deletion problems. *ACM Transactions on Algorithms (TALG)*, 16(4):1–38, 2020.
- [4] Akanksha Agrawal, Daniel Lokshantov, Amer E Mouawad, and Saket Saurabh. Simultaneous feedback vertex set: A parameterized perspective. *ACM Transactions on Computation Theory (TOCT)*, 10(4):1–25, 2018.
- [5] Mohsen Alambardar Meybodi, Fedor Fomin, Amer E Mouawad, and Fahad Panolan. On the parameterized complexity of $[1, j]$ -domination problems. In *FSTTCS 2018*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

- [6] Esther M Arkin, Aritra Banik, Paz Carmi, Gui Citovsky, Matthew J Katz, Joseph SB Mitchell, and Marina Simakov. Choice is hard. In *International Symposium on Algorithms and Computation*, pages 318–328. Springer, 2015.
- [7] Esther M Arkin, Aritra Banik, Paz Carmi, Gui Citovsky, Matthew J Katz, Joseph SB Mitchell, and Marina Simakov. Conflict-free covering. In *Conference on Computational Geometry*, page 17, 2015.
- [8] Esther M Arkin and Refael Hassin. Minimum-diameter covering problems. *Networks: An International Journal*, 36(3):147–155, 2000.
- [9] Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991.
- [10] Pranav Arora, Aritra Banik, Vijay Kumar Paliwal, and Venkatesh Raman. Some (in) tractable parameterizations of coloring and list-coloring. In *International Workshop on Frontiers in Algorithmics*, pages 126–139. Springer, 2018.
- [11] Vineet Bafna, Piotr Berman, and Toshihiro Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM Journal on Discrete Mathematics*, 12(3):289–297, 1999.
- [12] Brenda S Baker. Approximation algorithms for np-complete problems on planar graphs. *Journal of the ACM (JACM)*, 41(1):153–180, 1994.
- [13] Aritra Banik, Fahad Panolan, Venkatesh Raman, and Vibha Sahlot. Fréchet distance between a line and avatar point set. *Algorithmica*, 80(9):2616–2636, 2018.
- [14] Aritra Banik, Fahad Panolan, Venkatesh Raman, Vibha Sahlot, and Saket Saurabh. Parameterized complexity of geometric covering problems having conflicts. *Algorithmica*, Jul 2019.
- [15] B. Bergougnoux and M. M. Kanté. Fast exact algorithms for some connectivity problems parametrized by clique-width. *arXiv preprint arXiv:1707.03584*, 2017.

- [16] Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM Journal on Computing*, 39(2):546–563, 2009.
- [17] H. L. Bodlaender, S. Thomassé, and A. Yeo. Kernel bounds for disjoint cycles and disjoint paths. *Theoretical Computer Science*, 412(35):4570–4578, 2011.
- [18] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on computing*, 25(6):1305–1317, 1996.
- [19] Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Information and Computation*, 243:86–111, 2015.
- [20] Hans L. Bodlaender, Pål Gronås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michał Pilipczuk. A $\tilde{c}^k n$ 5-approximation algorithm for treewidth. *SIAM Journal on Computing*, 45(2):317–378, 2016.
- [21] Édouard Bonnet, Nick Brettell, O-joung Kwon, and Dániel Marx. Parameterized vertex deletion problems for hereditary graph classes with a block property. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 233–244. Springer, 2016.
- [22] Kellogg S. Booth and J. Howard Johnson. Dominating sets in chordal graphs. *SIAM J. Comput.*, 11:191–199, 1982.
- [23] Anudhyan Boral, Marek Cygan, Tomasz Kociumaka, and Marcin Pilipczuk. A Fast Branching Algorithm for Cluster Vertex Deletion. *Theory Comput. Syst.*, 58(2):357–376, 2016.
- [24] N. Boria, F. D. Croce, and V. T. Paschos. On the max min vertex cover problem. *Discrete Applied Mathematics*, 196:62–71, 2015.

- [25] Nicolas Bourgeois, Konrad K. Dabrowski, Marc Demange, and Vangelis Th Paschos. Playing with parameters: structural parameterization in graphs. *arXiv preprint arXiv:1309.6144*, 2013.
- [26] Lukasz Bożyk, Jan Derbisz, Tomasz Krawczyk, Jana Novotná, and Karolina Okrasa. Vertex deletion into bipartite permutation graphs. In *15th International Symposium on Parameterized and Exact Computation (IPEC 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.
- [27] Andreas Brandstädt. On robust algorithms for the maximum weight stable set problem. In *International Symposium on Fundamentals of Computation Theory*, pages 445–458. Springer, 2001.
- [28] Andreas Brandstadt, Jeremy P Spinrad, et al. *Graph classes: a survey*, volume 3. Siam, 1999.
- [29] Heinz Breu and David G. Kirkpatrick. Unit disk graph recognition is NP-hard. *Comput. Geom.*, 9(1-2):3–24, 1998.
- [30] Sharon Bruckner, Falk Hüffner, and Christian Komusiewicz. A graph modification approach for finding core–periphery structures in protein interaction networks. *Algorithms for Molecular Biology*, 10(1):1–13, 2015.
- [31] Jonathan F Buss and Judy Goldsmith. Nondeterminism within p. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 348–359. Springer, 1991.
- [32] Leizhen Cai. Fixed-Parameter Tractability of Graph Modification Problems for Hereditary Properties. *Inf. Process. Lett.*, 58(4):171–176, 1996.
- [33] Leizhen Cai. Parameterized complexity of vertex colouring. *Discrete Applied Mathematics*, 127(3):415–429, 2003.

- [34] Yixin Cao and Dániel Marx. Interval deletion is fixed-parameter tractable. *ACM Transactions on Algorithms (TALG)*, 11(3):1–35, 2015.
- [35] Yixin Cao and Dániel Marx. Chordal editing is fixed-parameter tractable. *Algorithmica*, 75(1):118–137, 2016.
- [36] Jianer Chen, Iyad A Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theoretical Computer Science*, 411(40-42):3736–3756, 2010.
- [37] Jianer Chen, Yang Liu, and Songjian Lu. An improved parameterized algorithm for the minimum node multiway cut problem. *Algorithmica*, 55(1):1–13, 2009.
- [38] Benny Chor, Mike Fellows, and David Juedes. Linear kernels in linear time, or how to save k colors in $o(n^2)$ steps. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 257–269. Springer, 2004.
- [39] Jayesh Choudhari, Anirban Dasgupta, Neeldhara Misra, and MS Ramanujan. Saving critical nodes with firefighters is fpt. In *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.
- [40] Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86(1-3):165–177, 1990.
- [41] Derek G. Corneil and Jean Fonlupt. The complexity of generalized clique covering. *Discrete Applied Mathematics*, 22(2):109–118, 1988.
- [42] B. Courcelle, J.A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33:125–150, 2000.
- [43] B. Courcelle and S. Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1):77–114, 2000.

- [44] Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and computation*, 85(1):12–75, 1990.
- [45] M. Cygan, F. V. Fomin, K. Lukasz, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.
- [46] M. Cygan and M. Pilipczuk. Split vertex deletion meets vertex cover: New fixed-parameter and exact exponential-time algorithms. *Inf. Process. Lett.*, 113(5-6):179–182, 2013.
- [47] Marek Cygan. Deterministic parameterized connected vertex cover. In *Scandinavian Workshop on Algorithm Theory*, pages 95–106. Springer, 2012.
- [48] Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as cnf-sat. *ACM Transactions on Algorithms (TALG)*, 12(3):41, 2016.
- [49] Marek Cygan and Marcin Pilipczuk. Split vertex deletion meets vertex cover: new fixed-parameter and exact exponential-time algorithms. *Information Processing Letters*, 113(5-6):179–182, 2013.
- [50] Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. On multiway cut parameterized above lower bounds. *TOCT*, 5(1):3:1–3:11, 2013.
- [51] Konrad K Dabrowski, Carl Feghali, Matthew Johnson, Giacomo Paesani, Daniël Paulusma, and Paweł Rzażewski. On cycle transversals and their connected variants in the absence of a small linear forest. *Algorithmica*, 82(10):2841–2866, 2020.
- [52] David P. Dailey. Uniqueness of colorability and colorability of planar 4-regular graphs are NP-complete. *Discrete Mathematics*, 30(3):289–293, 1980.

- [53] Andreas Darmann, Ulrich Pferschy, Joachim Schauer, and Gerhard J Woeginger. Paths, trees and matchings under disjunctive constraints. *Discrete Applied Mathematics*, 159(16):1726–1735, 2011.
- [54] Walter A. Deuber, Paul Erdős, David S. Gunderson, Alexandr V. Kostochka, and A. G. Meyer. Intersection Statements for Systems of Sets. *J. Comb. Theory, Ser. A*, 79(1):118–132, 1997.
- [55] Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
- [56] M. Dom, D. Lokshtanov, and S. Saurabh. Kernelization lower bounds through colors and ids. *ACM Transactions on Algorithms (TALG)*, 11(2):13, 2014.
- [57] Rodney G Downey and Michael R Fellows. *Fundamentals of parameterized complexity*, volume 4. Springer, 2013.
- [58] Rodney G Downey, Michael R Fellows, Catherine McCartin, and Frances Rosamond. Parameterized approximation of dominating set problems. *Information Processing Letters*, 109(1):68–70, 2008.
- [59] Leah Epstein, Lene M Favrhøldt, and Asaf Levin. Online variable-sized bin packing with conflicts. *Discrete Optimization*, 8(2):333–343, 2011.
- [60] Guy Even, Magnús M Halldórsson, Lotem Kaplan, and Dana Ron. Scheduling with conflicts: online and offline algorithms. *Journal of scheduling*, 12(2):199–224, 2009.
- [61] Martin Farber. On diameters and radii of bridged graphs. *Discrete Mathematics*, 73(3):249–260, 1989.
- [62] Michael Fellows and Frances Rosamond. The complexity ecology of parameters: an illustration using bounded max leaf number. In *Conference on Computability in Europe*, pages 268–277. Springer, 2007.

- [63] Michael R Fellows, Fedor V Fomin, Daniel Lokshtanov, Frances Rosamond, Saket Saurabh, Stefan Szeider, and Carsten Thomassen. On the complexity of some colorful problems parameterized by treewidth. *Information and Computation*, 209(2):143–153, 2011.
- [64] Michael R. Fellows, Bart M.P. Jansen, and Frances Rosamond. Towards fully multivariate algorithmics: Parameter ecology and the deconstruction of computational complexity. *European Journal of Combinatorics*, 34(3):541–566, 2013.
- [65] Fedor V. Fomin and Petr A. Golovach. Subexponential parameterized algorithms and kernelization on almost chordal graphs. In *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, pages 49:1–49:17, 2020.
- [66] Fedor V Fomin, Fabrizio Grandoni, and Dieter Kratsch. Solving connected dominating set faster than 2^n . *Algorithmica*, 52(2):153–166, 2008.
- [67] Fedor V. Fomin, Petteri Kaski, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Parameterized single-exponential time polynomial space algorithm for steiner tree. *SIAM J. Discret. Math.*, 33(1):327–345, 2019.
- [68] Fedor V Fomin and Dieter Kratsch. *Exact exponential algorithms*. Springer Science & Business Media, 2010.
- [69] Fedor V Fomin, Dieter Kratsch, and Gerhard J Woeginger. Exact (exponential) algorithms for the dominating set problem. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 245–256. Springer, 2004.
- [70] Fedor V Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar f -deletion: Approximation, kernelization and optimal fpt algorithms. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 470–479. IEEE, 2012.

- [71] Fedor V Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Efficient computation of representative families with applications in parameterized and exact algorithms. *Journal of the ACM (JACM)*, 63(4):29, 2016.
- [72] Fedor V. Fomin and Torstein JF Strømme. Vertex cover structural parameterization revisited. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 171–182. Springer, 2016.
- [73] Fedor V. Fomin, Ioan Todinca, and Yngve Villanger. Large induced subgraphs via triangulations and CMSO. *SIAM J. Comput.*, 44(1):54–87, 2015.
- [74] Fedor V Fomin and Yngve Villanger. Treewidth computation and extremal combinatorics. *Combinatorica*, 32(3):289–308, 2012.
- [75] Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct pcps for NP. *J. Comput. Syst. Sci.*, 77(1):91–106, 2011.
- [76] Robert Ganian, M. S. Ramanujan, and Stefan Szeider. Discovering archipelagos of tractability for constraint satisfaction and counting. *ACM Trans. Algorithms*, 13(2):29:1–29:32, 2017.
- [77] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H.Freeman and Company, 1979.
- [78] Michael R Garey, David S. Johnson, and Larry Stockmeyer. Some simplified np-complete graph problems. *Theoretical computer science*, 1(3):237–267, 1976.
- [79] Serge Gaspers and Stefan Szeider. Backdoors to satisfaction. In Hans L. Bodlaender, Rod Downey, Fedor V. Fomin, and Dániel Marx, editors, *The Multivariate Algorithmic Revolution and Beyond - Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*, volume 7370 of *Lecture Notes in Computer Science*, pages 287–317. Springer, 2012.

- [80] Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs*, volume 57. Elsevier, 2004.
- [81] Gregory Gutin, Diptapriyo Majumdar, Sebastian Ordyniak, and Magnus Wahlström. Parameterized pre-coloring extension and list coloring problems. *arXiv preprint arXiv:1907.12061*, 2019.
- [82] Michel Habib and Christophe Paul. A simple linear time algorithm for cograph recognition. *Discrete Applied Mathematics*, 145(2):183–197, 2005.
- [83] T. W. Haynes, S. Hedetniemi, and P. Slater. *Domination in graphs: advanced topics*. 1997.
- [84] Pinar Heggernes, Pim van 't Hof, Bart M. P. Jansen, Stefan Kratsch, and Yngve Villanger. Parameterized complexity of vertex deletion into perfect graph classes. *Theor. Comput. Sci.*, 511:172–180, 2013.
- [85] H.L.Bodlaender, E.J.V.Leeuwen, and J.M.Van Rooji. Faster algorithms on branch and clique decompositions. In *MFCSS*, pages 174–185. Springer, 2010.
- [86] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.
- [87] R. Impagliazzo and R. Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences*, 62:367–375, 2001.
- [88] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.
- [89] Yoichi Iwata, Yutaro Yamaguchi, and Yuichi Yoshida. 0/1/all CSPs, Half-integral A-path packing, and linear-time FPT algorithms. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 462–473. IEEE, 2018.

- [90] Pallavi Jain, Lawqueen Kanesh, and Pranabendu Misra. Conflict free version of covering problems on graphs: Classical and parameterized. In *International Computer Science Symposium in Russia*, pages 194–206. Springer, 2018.
- [91] Bart M. P. Jansen and Marcin Pilipczuk. Approximation and kernelization for chordal vertex deletion. *SIAM J. Discret. Math.*, 32(3):2258–2301, 2018.
- [92] Bart M.P. Jansen. *The power of data reduction: Kernels for fundamental graph problems*. PhD thesis, Utrecht University, 2013.
- [93] Bart M.P. Jansen and Hans L. Bodlaender. Vertex cover kernelization revisited. *Theory of Computing Systems*, 53(2):263–299, 2013.
- [94] Bart MP Jansen and Jari JH de Kroon. Fpt algorithms to compute the elimination distance to bipartite graphs and more. *arXiv preprint arXiv:2106.04191*, 2021.
- [95] Bart MP Jansen and Stefan Kratsch. Data reduction for graph coloring problems. *Information and Computation*, 231:70–88, 2013.
- [96] Bart MP Jansen, Daniel Lokshtanov, and Saket Saurabh. A near-optimal planarization algorithm. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1802–1811. SIAM, 2014.
- [97] Bart M.P. Jansen, Venkatesh Raman, and Martin Vatshelle. Parameter ecology for feedback vertex set. *Tsinghua Science and Technology*, 19(4):387–409, 2014.
- [98] David S Johnson, Mihalis Yannakakis, and Christos H Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988.
- [99] Viggo Kann. Polynomially bounded minimization problems which are hard to approximate. In *International Colloquium on Automata, Languages, and Programming*, pages 52–63. Springer, 1993.

- [100] Ken-ichi Kawarabayashi and Anastasios Sidiropoulos. Polylogarithmic approximation for minimum planarization (almost). In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 779–788. IEEE, 2017.
- [101] Eun Jung Kim and O-joung Kwon. Erdős-pósa property of chordless cycles and its applications. *J. Comb. Theory, Ser. B*, 145:65–112, 2020.
- [102] Eun Jung Kim, Alexander Langer, Christophe Paul, Felix Reidl, Peter Rossmanith, Ignasi Sau, and Somnath Sikdar. Linear kernels and single-exponential algorithms via protrusion decompositions. *ACM Transactions on Algorithms (TALG)*, 12(2):1–41, 2015.
- [103] Ton Kloks. *Treewidth: computations and approximations*, volume 842. Springer Science & Business Media, 1994.
- [104] Dušan Knop, Martin Koutecký, Tomáš Masařík, and Tomáš Toufar. Simplified algorithmic metatheorems beyond mso: Treewidth and neighborhood diversity. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 344–357. Springer, 2017.
- [105] Dusan Knop, Tomáš Masařík, and Tomáš Toufar. Parameterized complexity of fair deletion problems II. *CoRR*, abs/1803.06878, 2018.
- [106] Tomasz Kociumaka and Marcin Pilipczuk. Faster deterministic Feedback Vertex Set. *Inf. Process. Lett.*, 114(10):556–560, 2014.
- [107] Stefan Kratsch and Magnus Wahlström. Two edge modification problems without polynomial kernels. *Discrete Optimization*, 10(3):193–199, 2013.
- [108] V. S. Anil Kumar, Sunil Arya, and H. Ramesh. Hardness of set cover with intersection 1. In Ugo Montanari, José D. P. Rolim, and Emo Welzl, editors, *Automata, Languages and Programming*, pages 624–635, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

- [109] C Lekkeikerker and J Boland. Representation of a finite graph by a set of intervals on the real line. *Fundamenta Mathematicae*, 51(1):45–64, 1962.
- [110] John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is np-complete. *J. Comput. Syst. Sci.*, 20(2):219–230, 1980.
- [111] Mathieu Liedloff, Pedro Montealegre, and Ioan Todinca. Beyond classes of graphs with "few" minimal separators: FPT results through potential maximal cliques. *Algorithmica*, 81(3):986–1005, 2019.
- [112] Lishin Lin and Sartaj Sahni. Fair edge deletion problems. *IEEE transactions on computers*, 38(5):756–761, 1989.
- [113] Daniel Lokshantov. Wheel-free deletion is $W[2]$ -hard. In *Parameterized and Exact Computation, Third International Workshop, IWPEC 2008, Victoria, Canada, May 14-16, 2008. Proceedings*, pages 141–147, 2008.
- [114] Daniel Lokshantov, Dániel Marx, and Saket Saurabh. Slightly superexponential parameterized problems. *SIAM J. Comput.*, 47(3):675–702, 2018.
- [115] Daniel Lokshantov, Pranabendu Misra, Fahad Panolan, and Saket Saurabh. Deterministic truncation of linear matroids. *ACM Transactions on Algorithms (TALG)*, 14(2):14, 2018.
- [116] Daniel Lokshantov, Pranabendu Misra, MS Ramanujan, and Saket Saurabh. Hitting selected (odd) cycles. *SIAM Journal on Discrete Mathematics*, 31(3):1581–1615, 2017.
- [117] Daniel Lokshantov, NS Narayanaswamy, Venkatesh Raman, MS Ramanujan, and Saket Saurabh. Faster parameterized algorithms using linear programming. *ACM Transactions on Algorithms (TALG)*, 11(2):1–31, 2014.
- [118] Daniel Lokshantov, Fahad Panolan, Saket Saurabh, Roohani Sharma, and Meirav Zehavi. Covering small independent sets and separators with applications to pa-

- parameterized algorithms. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2785–2800. Society for Industrial and Applied Mathematics, 2018.
- [119] Daniel Lokshantov and MS Ramanujan. Parameterized tractability of multiway cut with parity constraints. In *International Colloquium on Automata, Languages, and Programming*, pages 750–761. Springer, 2012.
- [120] Daniel Lokshantov, MS Ramanujan, and Saket Saurabh. A linear time parameterized algorithm for directed feedback vertex set. *arXiv preprint arXiv:1609.04347*, 2016.
- [121] Daniel Lokshantov, MS Ramanujan, Saket Saurabh, and Meirav Zehavi. Reducing CMSO Model Checking to Highly Connected Graphs. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
- [122] Diptapriyo Majumdar. *Classical and Approximate Kernels for Structural Parameterizations of some Graph Parameters*. PhD thesis, HOMI BHABHA NATIONAL INSTITUTE, 2018.
- [123] Diptapriyo Majumdar and Venkatesh Raman. Structural parameterizations of undirected feedback vertex set: FPT algorithms and kernelization. *Algorithmica*, 80(9):2683–2724, 2018.
- [124] Diptapriyo Majumdar, Venkatesh Raman, and Saket Saurabh. Polynomial kernels for vertex cover parameterized by small degree modulators. *Theory Comput. Syst.*, 62(8):1910–1951, 2018.
- [125] Kazuhisa Makino and Takeaki Uno. New algorithms for enumerating all maximal cliques. In *Scandinavian Workshop on Algorithm Theory*, pages 260–272. Springer, 2004.

- [126] Dániel Marx. Parameterized graph separation problems. *Theoretical Computer Science*, 351(3):394–406, 2006.
- [127] Dániel Marx. A parameterized view on matroid optimization problems. *Theoretical Computer Science*, 410(44):4471–4479, 2009.
- [128] Dániel Marx, Barry O’Sullivan, and Igor Razgon. Treewidth reduction for constrained separation and bipartization problems. In *27th International Symposium on Theoretical Aspects of Computer Science-STACS 2010*, pages 561–572, 2010.
- [129] Dániel Marx, Ario Salmasi, and Anastasios Sidiropoulos. Constant-factor approximations for asymmetric tsp on nearly-embeddable graphs. In *LIPICs-Leibniz International Proceedings in Informatics*, volume 60. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.
- [130] Tomáš Masařík and Tomáš Toufar. Parameterized complexity of fair deletion problems. In T.V. Gopal, Gerhard Jäger, and Silvia Steila, editors, *Theory and Applications of Models of Computation*, pages 628–642, Cham, 2017. Springer International Publishing.
- [131] George J Minty. On maximal independent sets of vertices in claw-free graphs. *Journal of Combinatorial Theory, Series B*, 28(3):284–304, 1980.
- [132] Neeldhara Misra, Geevarghese Philip, Venkatesh Raman, and Saket Saurabh. On parameterized independent feedback vertex set. *Theoretical Computer Science*, 461:65–75, 2012.
- [133] Neeldhara Misra, Geevarghese Philip, Venkatesh Raman, Saket Saurabh, and Somnath Sikdar. Fpt algorithms for connected feedback vertex set. *Journal of Combinatorial Optimization*, 24(2):131–146, 2012.

- [134] M.R.Fellows, B.M.P.Jansen, and F.A.Rosamond. Towards fully multivariate algorithmics: Parameter ecology and the deconstruction of computational complexity. *European Journal of Combinatorics*, 34(3):541–566, 2013.
- [135] Jaroslav Nešetřil and Patrice Ossona De Mendez. On nowhere dense graphs. *European Journal of Combinatorics*, 32(4):600–617, 2011.
- [136] Sang-il Oum, Sigve Hortemo Sæther, and Martin Vatshelle. Faster algorithms parameterized by clique-width. *arXiv preprint arXiv:1311.0224*, 2013.
- [137] James G Oxley. *Matroid theory*, volume 3. Oxford University Press, USA, 2006.
- [138] Daniël Paulusma. Open problems on graph coloring for special graph classes. In *Graph-Theoretic Concepts in Computer Science - 41st International Workshop, WG 2015, Garching, Germany, June 17-19, 2015, Revised Papers*, pages 16–30, 2015.
- [139] Ulrich Pferschy and Joachim Schauer. The maximum flow problem with conflict and forcing conditions. In *Network Optimization*, pages 289–294. Springer, 2011.
- [140] Ulrich Pferschy and Joachim Schauer. Approximation of knapsack problems with conflict and forcing graphs. *Journal of Combinatorial Optimization*, 33(4):1300–1323, 2017.
- [141] Geevarghese Philip, Ashutosh Rai, and Saket Saurabh. Generalized pseudoforest deletion: Algorithms and uniform kernel. *SIAM J. Discrete Math.*, 32(2):882–901, 2018.
- [142] Vijay Raghavan and Jeremy P. Spinrad. Robust algorithms for restricted domains. *J. Algorithms*, 48(1):160–172, 2003.
- [143] Ashutosh Rai and M. S. Ramanujan. Strong parameterized deletion: Bipartite graphs. In *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016*, pages 21:1–21:14, 2016.

- [144] Ashutosh Rai and Saket Saurabh. Bivariate complexity analysis of almost forest deletion. *Theor. Comput. Sci.*, 708:18–33, 2018.
- [145] Venkatesh Raman and Saket Saurabh. Short cycles make w-hard problems hard: Fpt algorithms for w-hard problems in graphs with no short cycles. *Algorithmica*, 52(2):203–225, 2008.
- [146] Romeo Rizzi. Minimum weakly fundamental cycle bases are hard to find. *Algorithmica*, 53(3):402–424, 2009.
- [147] Neil Robertson and Paul D. Seymour. Graph minors. xiii. the disjoint paths problem. *Journal of combinatorial theory, Series B*, 63(1):65–110, 1995.
- [148] Jeremy P. Spinrad. *Efficient graph representations*. American Mathematical Society, 2003.
- [149] Jan Arne Telle. Complexity of domination-type problems in graphs. *Nord. J. Comput.*, 1(1):157–171, 1994.
- [150] René van Bevern, Oxana Yu Tsidulko, and Philipp Zschoche. Fixed-parameter algorithms for maximum-profit facility location under matroid constraints. In *Proceedings of the 11th International Conference on Algorithms and Complexity*, 2019.
- [151] Johan MM Van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In *Algorithms - ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, pages 566–577, 2009.
- [152] Pim Van’t Hof and Yngve Villanger. Proper interval vertex deletion. *Algorithmica*, 65(4):845–867, 2013.
- [153] Mihalis Yannakakis. Node-deletion problems on bipartite graphs. *SIAM J. Comput.*, 10(2):310–327, 1981.

- [154] M. Zehavi. Maximum minimal vertex cover parameterized by vertex cover. *SIAM Journal of Discrete Mathematics*, 31(4):2440–2456, 2017.